Accuracy and Efficiency Comparison of Object Detection Open-Source Models

https://doi.org/10.3991/ijoe.v17i05.21833

Brahim Jabir (✉), Noureddine Falih, Khalid Rahmani
Sultan Moulay Slimane University, Beni Mellal, Morocco
ibra.jabir@gmail.com

Abstract—In agriculture, weeds cause direct damage to the crop, and it primarily affects the crop yield potential. Manual and mechanical weeding methods consume a lot of energy and time and do not give efficient results. Chemical weed control is still the best way to control weeds. However, the widespread and large-scale use of herbicides is harmful to the environment. Our study's objective is to propose an efficient model for a smart system to detect weeds in crops in real-time using computer vision. Our experiment dataset contains images of two different weed species well known in our region strained in this region with a temperate climate. The first is the Phalaris Paradoxa. The second is Convolvulus, manually captured with a professional camera from fields under different lighting conditions (from morning to afternoon in sunny and cloudy weather). The detection of weed and crop has experimented with four recent pre-configured open-source computer vision models for object detection: Detectron2, EfficientDet, YOLO, and Faster R-CNN. The performance comparison of weed detection models is executed on the Open CV and Keras platform using python language.

Keywords—Precision agriculture, Weed Identification, Deep learning, Object detection, Open cv

1 Introduction

Object detection is one of the most active fields of research in computer vision, where it involves both object classification, classifying every object in the image, and object localization [1]. Agriculture is a field affected by these innovations to promote production and guarantee food security [2]. Therefore, the thinking on a set of more developed systems based on the detection of objects in real-time, such as autonomous robotics for weed spraying, livestock detection, and vehicle safety, has become necessary. In our case, the focus is on the identification of weed from crop because of its great importance in precision farming, as weed act as a pest to crop and competes for space, nutrients, water, light and hinders the growth of crops in the field. The conventional way of eliminating weed is to spray herbicides or manual plucking [3]. The manual weed removal method is a tedious task, as it needs vast labor work. Usage of herbicides harms the health of living beings and the surrounding environment.

Hence, there is an urge to automate the process of weed identification. An approach based on deep learning is proposed to differentiate crop from weeds in real time to contribute to a localized spraying system so as not to distribute the herbicides on a large scale. The approaches based on convolutional neural networks are, to date, the most efficient models, and thanks to transfer learning, we can use a pre-existing model trained on a vast dataset for our tasks [4]. Consequently, reducing the cost of training new deep learning models, and since the datasets have been vetted, we can be assured of the quality. Therefore, we try to train four recent models in our custom data to know the important factors for obtaining the best results and choose the best of theme in terms of accuracy and efficiency. In addition, try to implement it on an intelligent system based on Raspberry [5].

This article will define the architecture of the four open-source models, Detectron2, EfficientDet, YOLO, and Faster R-CNN. We will apply these models based on images collected directly from fields, and then we will compare them in terms of precision and error to choose the best. For this, we will work with the Tensorflow and Keras libraries for learning and object detection [6]. To improve the models' performance, we will use some efficient techniques such as data augmentation and propose methods to speed up our model. In the final section, we make a comparison between these sets of methods in accuracy and speed.

2 Research Method

In this section, we will define the Methods, Software, hardware, and libraries used in the experiment, passing through the dataset necessary for our models' training.

2.1 Tensorflow

In November 2015, Google developed a new programming framework (framework) for numerical computation, called TensorFlow, and made it open source. This framework is intended, particularly for machine learning and artificial intelligence technologies, from which its "Tensor" nomination was inspired. Tensors are multi-dimensional data tables that manage common operations on neural networks. It is a Matrix. Gmail, Google Photos, and Voice Recognition can be used as an illustration of the TensorFlow application [7].

2.2 Keras

A very intuitive library of Deep learning in python, or what François Chollet called, in 2017, Keras. This high-level neural networks API allows you to create and train deep learning models. Thus, it is used in rapid prototyping, advanced research, and going into production. Keras aims to go from idea to result in a short time, under the ONEIROS project (open-ended Neuro-Electronic Intelligent Robot Operating system). However, as its founder explained, Keras is designed as an interface that presents a higher-level, more intuitive set of abstractions that make it easy to configure neural networks independent of the back-end computer library [8].

2.3 Python

The Python programming language is widely used in deep learning. Its code does not need to be compiled to work. It is a high-level language interpreted and oriented, allowing a reasonable reduction of the cost of maintaining the codes and encouraging them the modality and the reusability. The Pythons libraries are Free for the majority of platforms [9].

2.4 Hardware configuration used in the implementation

An efficient implementation of an Object Detection Models will be valid according to precise hardware conditions. For our case, we used an HP i7 CPU 2.40 GHZ laptop pc, with Nvidia GeForce GT525M graphics card, a size of 8 GB concerning the RAM, and hard disk of size 500GB.

2.5 Object detection methods based on deep learning

For our study, we will exploit our models' power thanks to transfer learning methods, which allow us to use the learning acquired on a general classification problem to apply it again to a particular problem [10]. Research teams specializing in improving CNNs provide these models. They publish their technical innovations, as well as the details of the networks trained on reference databases. For example, the COCO test-dev set, ImageNet challenge (ILSVRC), can arrange object detectors in two main categories; Two-stage detectors, such as Faster R-CNN, have a region-of-interest proposal step and another final classification and bounding-box regression of objects taking these regions as input. Single-Stage Detectors such as YOLO, EfficiendDet, Detectron2[11] consider object detection a simple regression problem learning the class probabilities and bounding box coordinates from input images.

EfficientDet: A new version of Efficientnet, EfficientDet, is a model for real-time object detection via a personalized detection and classification network. It is smaller, weighing 17MB. It is an open-source neural network model for the image detection computer vision task. However, it is published initially in the TensorFlow and Keras platforms, then in Pytorch. EfficientDet versions vary from EfficientDet B0 to B7[12], the last version of EfficientDet-D7 achieves 55.1 [11].

Faster R-CNN: In 2014, Ross Girschick et al. invented a method of object detection named R-CNN series and improved with faster R-CNN [13]. This is a more precise method. It is a two-step deep learning object finder, namely:

- Identification of regions of interest
- Transmission to a convolutional neural network

Classification is done through the transmission of the produced feature maps to an SVM support vector machine, thus, the computation of the regression between the predicted bounding boxes and the ground truth bounding boxes. Faster R-CNN has an AP of 34.9 to 36.8 on test-dev coco [14].

YOLOv5: Yolo is an "acronym referring to the English expression you only look once"1. It is a system addressed to detect objects in real-time from the images at 30 fps. It has had an improved AP (49.2 to 50.1) on test-dev coco. Yolov5 is a continuation of the recent versions of the YOLO series. It is smaller and generally more comfortable to use in training. Changing its architecture and exporting to many deployment environments is just as easy. This series, published by Glenn Jocher on June 9, 2020, is implemented in PyTorch [15].

Detectron2: The detection of key points, the detection of objects, and the semantic segmentation are part of the priority area of interest of Detectron2. It is a system written in Pytorch and contains improved versions of faster R47-CNN, Mask R-CNN, Retinanet, and Denspose. Likewise, it does support the rapid implementation and evaluation of new computer vision research, which is why it was invented by Facebook AI Research (FAIR). Its AP can reach 64.31 on COCO test-dev [16].

Below we illustrate the results of the four models object of our study (Detectron2, EfficientDet, YOLOv5, Faster R-CNN) on COCO test-dev in terms of Average Precision (AP) chosen as the primary evaluation metric, which averages AP across IoU thresholds from 0.5 to 0.95 with an interval of 0.05. The data illustrates on graphs represents the highest and lowest precision value for each version of each model (figure 1).

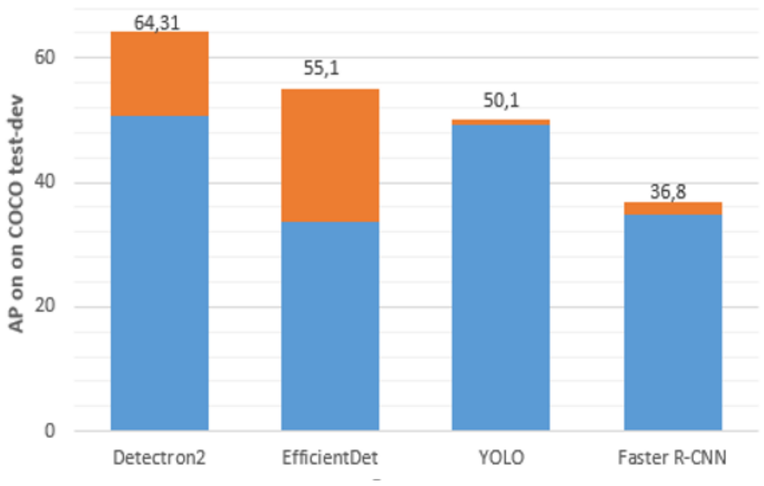

Fig. 1. Accuracy comparison on COCO test-dev

2.6 Dataset preparation

The preparation and configuration of the dataset among the most important phases in the process of object detection. The dataset plays a significant role in scientific research. They have been one of the most important factors for them for progress in deep learning [17]. Unfortunately, data is still difficult and more expensive to generate and annotate. A set of operations, as summarized in the figure 2 below, are necessary in preparing the dataset, and we will detail them in this section.

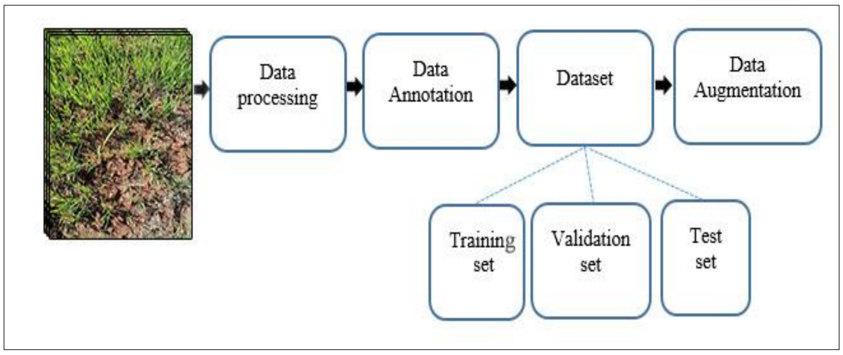

Fig. 2. Dataset preparation

An experiment was done in this regard in the region of BeniMellal-Khenifra in Morocco, known for its temperate climate. It is a question of collecting real images of two types of weeds. The first is the "Phalaris Paradoxa" the second is "Convolvulus" (figure 3, 4), belong to two different classes. For this experiment, a professional Nikon 7000 camera is used to capture 2000 images of wheat fields under several lighting conditions (sunny and cloudy weather from morning to evening).

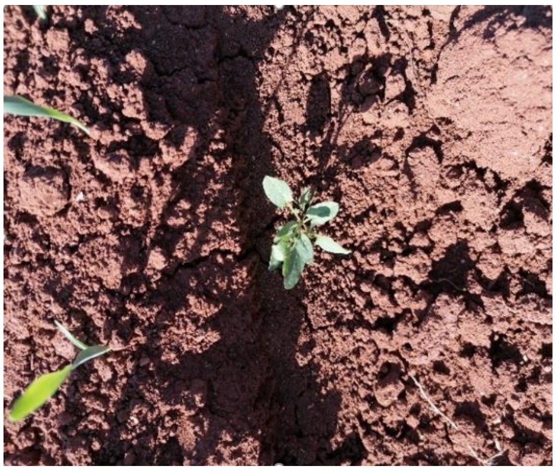

Fig. 3. Convolvulus weed

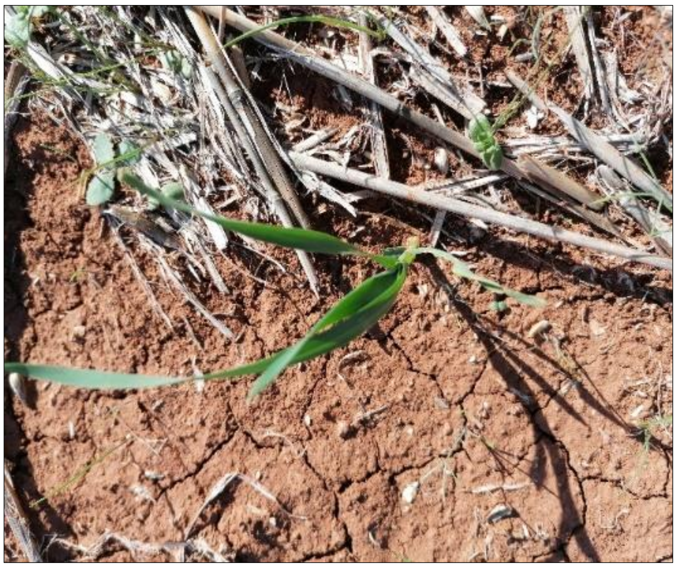

Fig. 4. Phalaris weed

These RGB images show the two types of plants at different stages of growth; they are in png format and of various sizes, which requires us to resize them before being used as input to the CNN model in the form 416x416. Then the annotation phase comes to assign a legend to the objects containing in the image. This technique creates training data for computer vision to train our model to learn how to see an item as we do [18]. To do this, we used Use LabelImg Annotations (figure 5), which is based on the bounding box to force the labels to draw a box as close as possible to the edges of the key objects in the image (weeds), the points forming the frames (at the top left and in the bottom right) are stored in an XML file. Then we divided the images into train, validation and test splits to avoid the model being overfitted and to evaluate the model with metrics generated for this purpose. The extended part of our dataset (70%) is the training set reserved for training our models. Accuracy resulted in these images after the training step will be taken to memorize the right output. Validation's second set is a separate section of the dataset (20%) used during training to evaluate our model's performance reporting the validation metrics continually after each training epoch, such as average precision. Finally, 10% of our dataset was used as a test set after the training experiments to get an idea of the model's final performance [19].

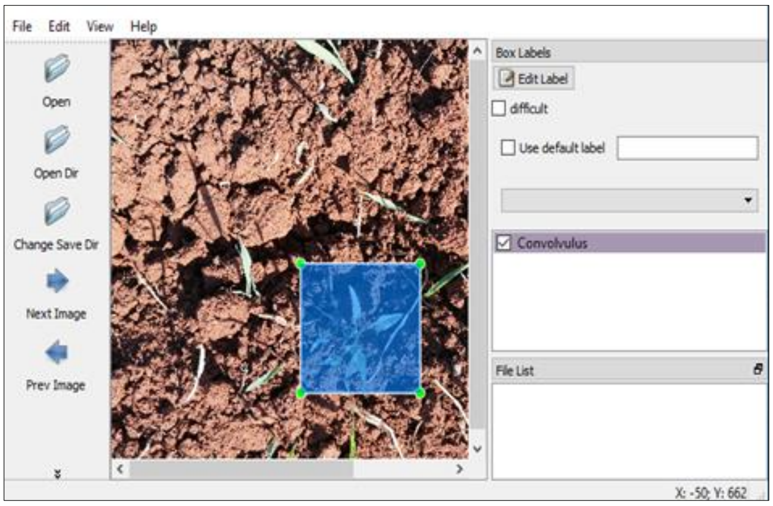

Fig. 5. Label object bounding boxes in images

Besides resizing input images, there are other preprocessing options to ensure that the dataset images are correctly formatted for our model to detect edges efficiently. This processing applies to all images (train, valid, and test) to reduce learning time and improve inference speed [20]. In our case, we used a set of options, and here some examples:

- Static Crop: Crop each image to the specified section, such as the bottom third.
- Auto-Adjust Contrast: Boosts contrast based on the image's histogram to improve normalization and line detection in varying lighting conditions.
- Auto-Orient: Discard EXIF rotations and standardize pixel ordering.

Data Augmentation is the last process in preparing the dataset. Its options are applied to images in our training set random to produce more training data and increase our model accuracy. This way applies domain-specific techniques to examples to generate more training data [21]. These methods give us about 3000 images in total after augmentation. Below examples of some methods used in the experiment:

- **Flips:** Haphazardly reflecting a picture about its x or y pivot drives our model to perceive that an image need not generally be perused from left to right or up to down. This method helps the model to be obtuse toward image orientation (figure 6, 7).

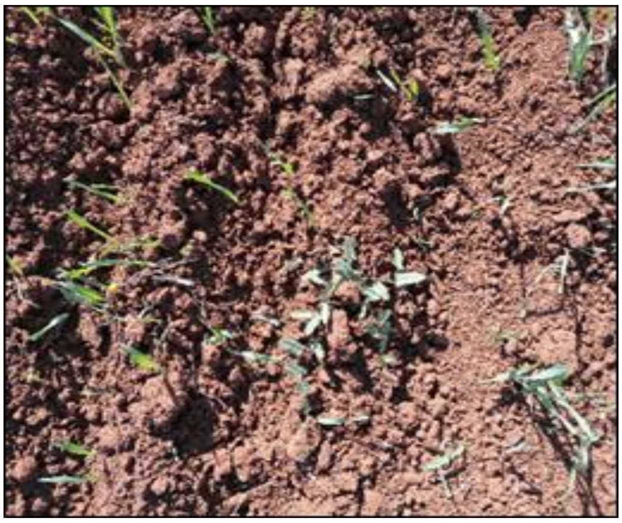

Fig. 6. Image preprocessed

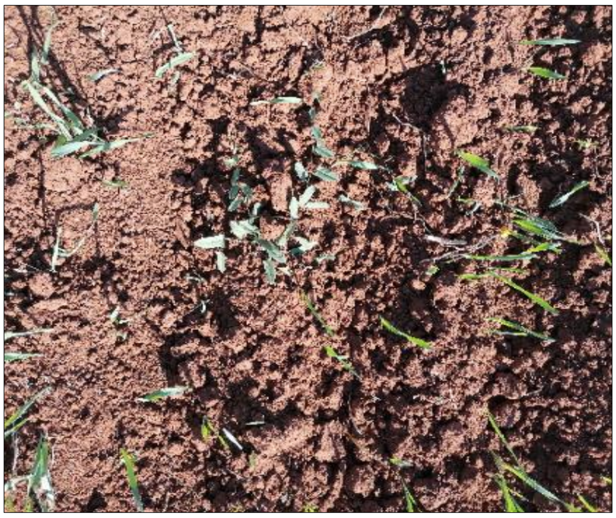

Fig. 7. Horizontal flip of image

- **Rotations :**Turning a picture is especially significant when a model might be utilized in a non-fixed position, similar to our intelligent system that will be proposed. These techniques assist the model with being resilient to camera roll and distinguishing objects well even when the camera or subject is not entirely adjusted(figure 8).

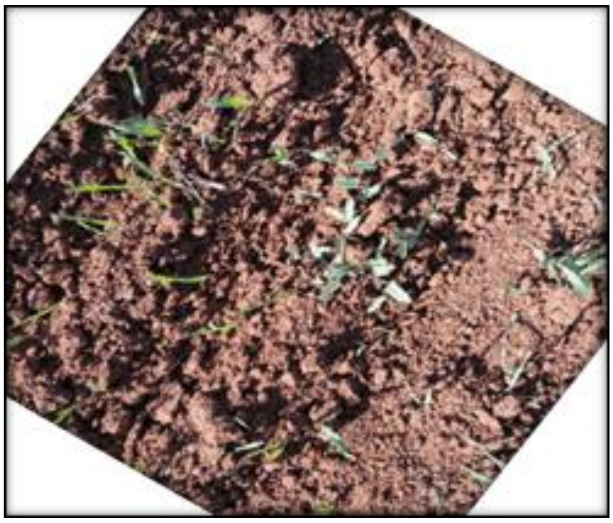

Fig. 8. New image resulted after 34° of rotation

- **Exposure:** It's critical to consider the most extreme and least of brightness in the fields, so this method Adds inconstancy to picture brightness and adjusting the picture to be arbitrarily brighter and darker to assist our model with being versatile to lighting and camera setting changes (figure 9, 10).

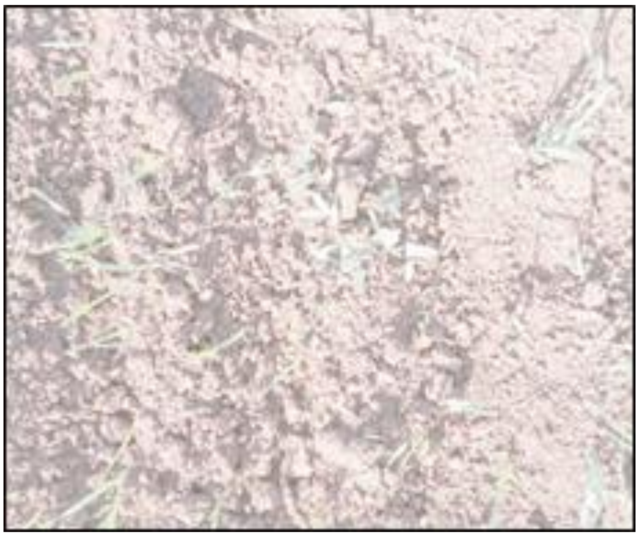

Fig. 9. New image generated with 50%

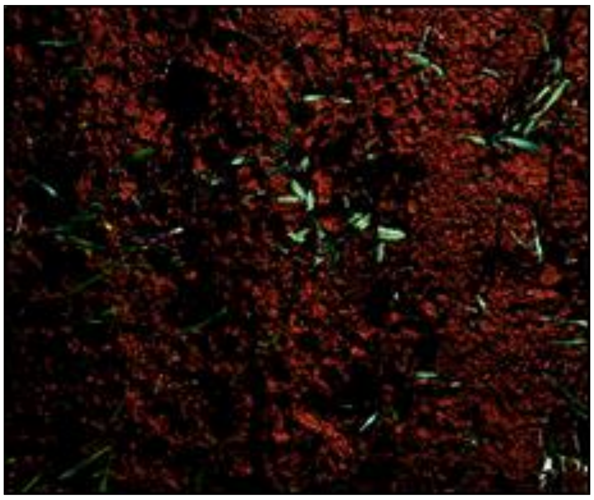

Fig. 10. New image generated with -50% of brightness

- **Noise:** This technique adds noise to assist your model with being tough to camera artifacts. A typical method is "salt and pepper noise," wherein picture pixels are haphazardly changed over to be totally dark or white (figure 11).

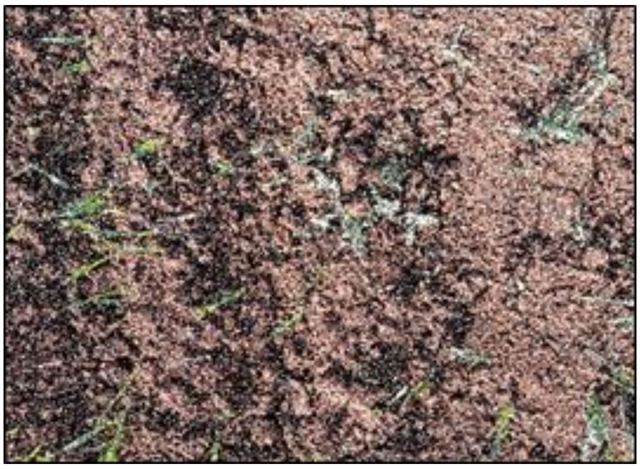

Fig. 11. New noisy images generated

3 Results and Discussion

During our experiments, we implemented the four models to train them to our dataset. The configuration of these models differs from one to the other. Each object detection model has a shell of a preparation setup explicit to each one, given by the creators (base pipeline file). And a pretrained checkpoint indicates the file of pretrained weights saved from when the model was trained on an enormous dataset. We also specify some training parameters like input image size number of training epochs and steps. There is several metrics to evaluate the performance of the neural networks and deep learning models after training them on our dataset, we use the precision metrics and also calculating the error. The precision gauges the model's precision in arranging a images as positive. The precision is determined as the proportion between the quantity of Positive images correctly arranged (True positive) to the complete number of picture named Positive (either correctly or incorrectly) (true positive + false positive) (1) [22]. We also evaluate models accuracy by running inference on the test dataset to show predictions on them [23].

$$\text{Precision} = \frac{TP}{TP+FP} \tag{1}$$

TP: True positive
FP: False positive

To display the results obtained for the four models, we illustrate in what follows, the results in terms of precision and error for each of the four models.

3.1 Results obtained for the EfficientDet model

After training the model for 3500 steps, we get results like the accompanying in TensorBoard:

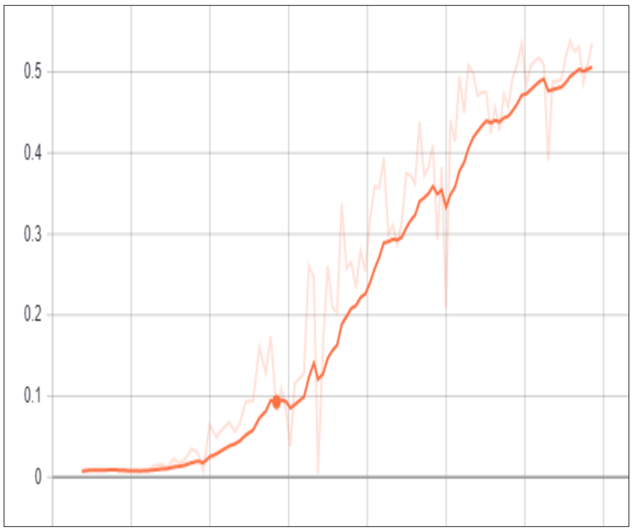

Fig. 12. Precision of the EfficientDet model

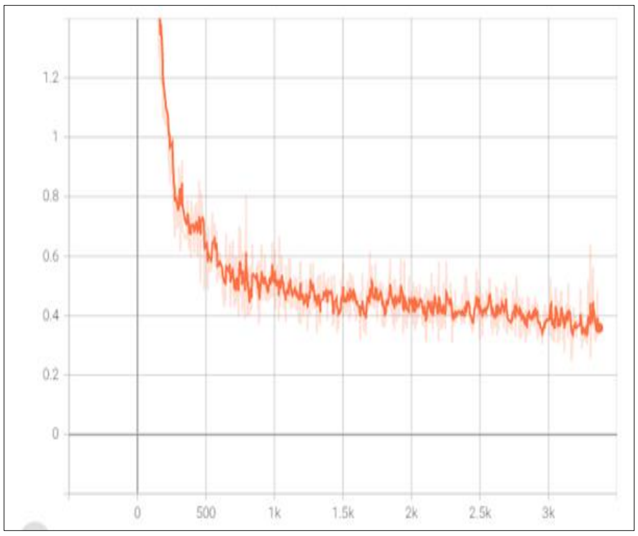

Fig. 13. Error of the EfficiendDet model

The results obtained, the following remarks are noted: According to Figure 12, the accuracy of the training increases with each stage to reach 52% after 12 hours of training. This reflects that with each progression, the model learns more data. So we need more steps to get more accuracy. The training error decreases with the number of epochs, reaching 38% (figure 13). Despite the medium precision of EfficientDet training, we had some interesting predictions on the test images (figure 14). This shows that we only need to speed up the performance by modifying specific parameters, such as, the number of training steps, the data-augmentation options on the dataset.

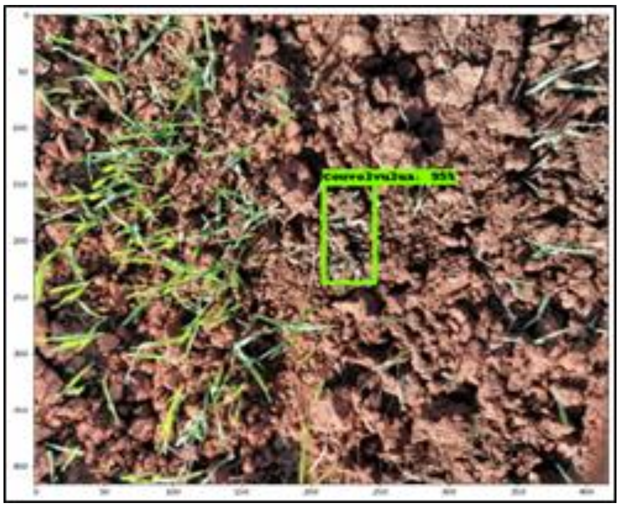

Fig. 14. Weed prediction on test image

3.2 Results obtained for the YOLOv5 model

After the end of the YOLOv5 training on 200 epochs, we evaluate its performance by the precision and error metrics on Tensorboard as follow:

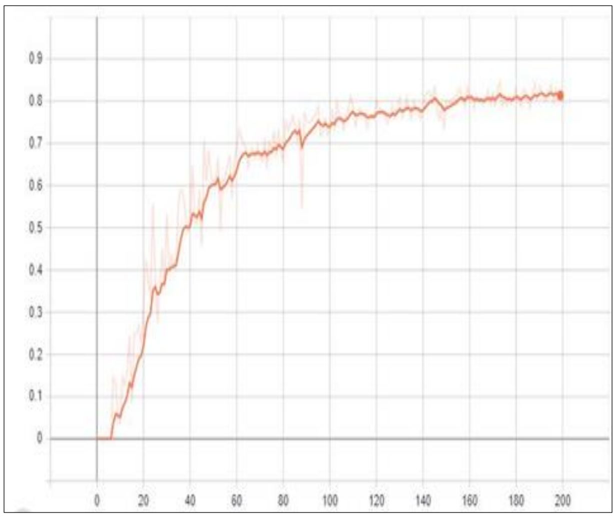

Fig. 15. The precision of the YOLOv5 model

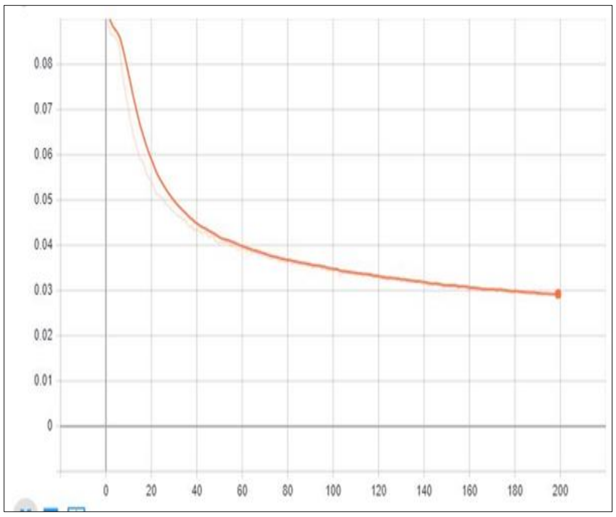

Fig. 16. Error of the YOLOv5 model

From the results below, we can see that the performance metrics are saved to Tensorboard, and showed an interesting precision reached 82% (figure 15). A total error has decreased to 30% (figure 16). The YOLOv5 training is faster than the other models, approximately 2 hours of training. It means that the performance of the model is always able to improve with each epoch. The predictions on the test images gave the same fact (figure 17), most of the predictions are correct with a rate of 98%, here is a model of these predictions:

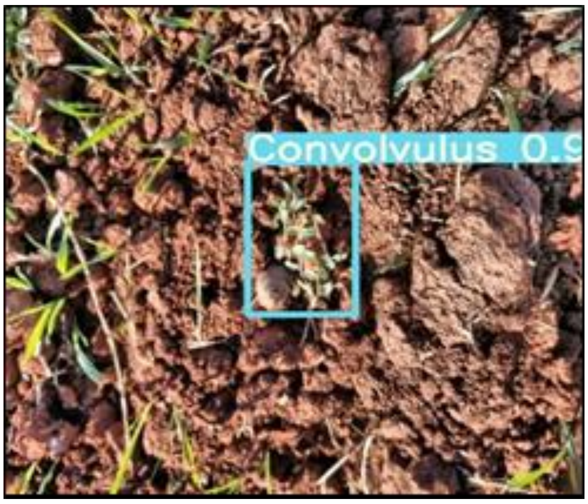

Fig. 17.YOLOv5 prediction on test image

3.3 Results obtained for the Detectron2 model

We display results on Tensorboard to see how the training procedure has performed on 1500 steps. There are a lot of metrics of interest in this evaluation tool. We focus on the total of loss and accuracy; as shown in the figures below:

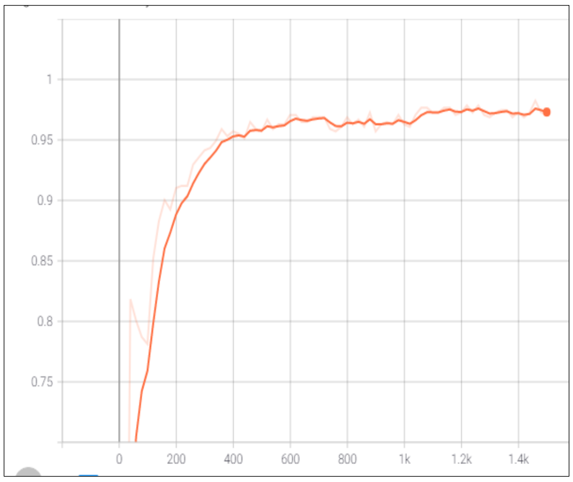

Fig. 18. Detectron2 Accuracy

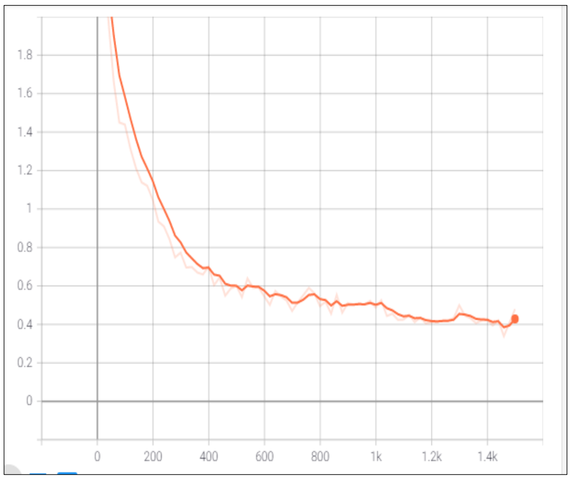

Fig. 19. Detectron2 Error

After analyzing the results acquired, the accompanying comments can be seen: According to the figure 18, the training accuracy increments with the number of steps. This reflects that with each step, the model learns more information. We ought to change up steps if Val accuracy is as yet rising, change down if overfit. The training has arrived at 97% on the last steps, after 5 hours of training that is interesting. We also notice that all of the misclassified images have an error rate of 42.12% (figure 19). Finally, we can run our new custom Detectron2 detector on real images that the model has never seen. We get an interesting forecast demonstrating that the model has learned how to identify weeds from the crop (figure 20).

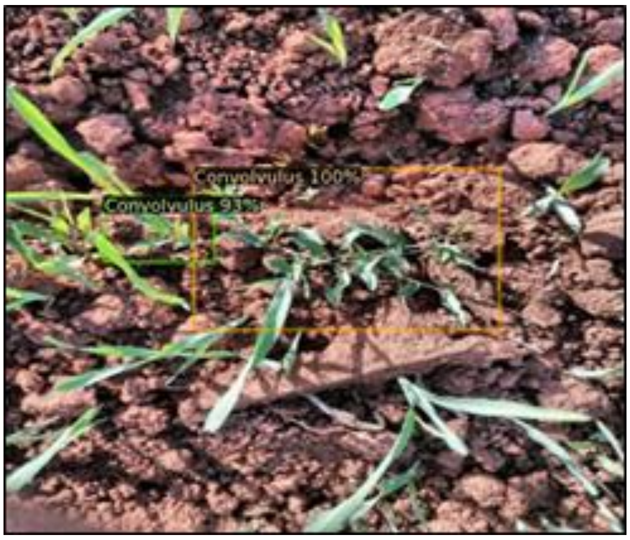

Fig. 20. Detectron2 inference on test images

3.4 Results obtained for the Faster R-CNN model

The results displayed on the Tensorboard tool after 10000 steps show the following metrics:

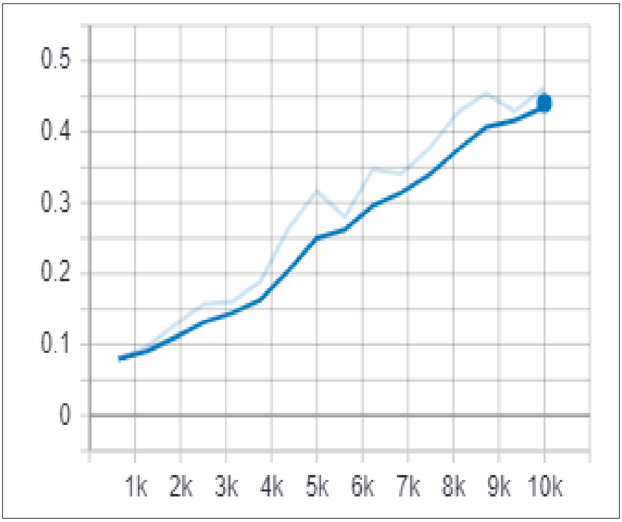

Fig. 21. The accuracy of Faster R-CNN

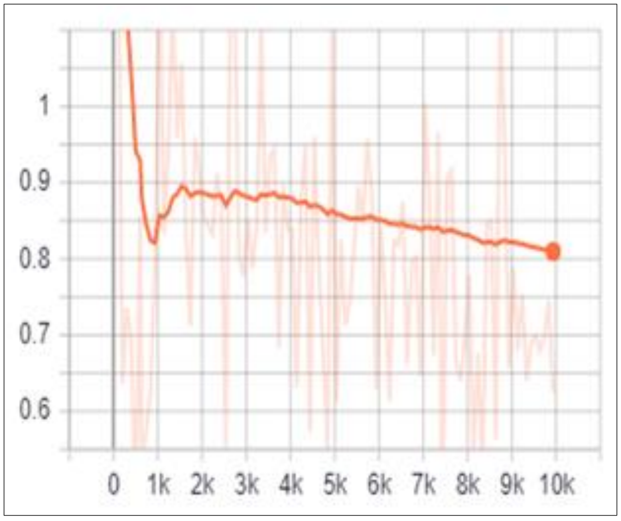

Fig. 22. The accuracy of Faster R-CNN

The figures above show that learning accuracy increases with each step to reach 45% after 10 hours (figure 21). This reflects the number of epochs is an essential factor where the model learns more information. Although the number of steps is high compared to other models, the accuracy is average. Likewise, the error changes with the progress of learning. It increases and decreases. In the last stages, it only reaches 80% (figure 22). Despite all that, the predictions given on the test images show different results on test images as it shows in the figure below (figure 23), it shows that it is necessary to improve the model by dragging it on more steps and generate more picture with the data augmentation tool.

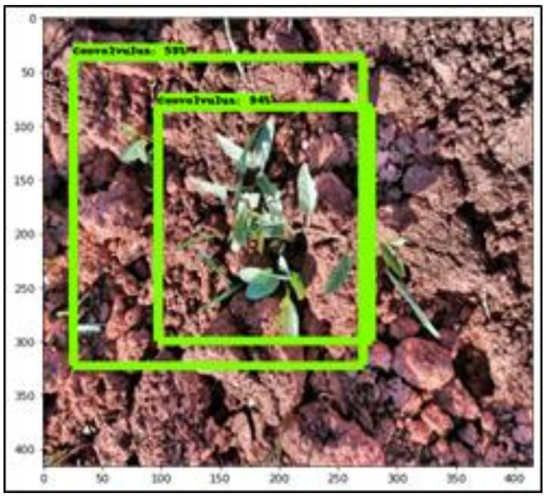

Fig. 23. Inference on test images

After analyzing the results obtained on all open-source object detection models, we compare the results of the different detectors in terms of both accuracy and error. For this purpose, the results achieved by these detectors on our custom data available in table 1.

Table 1. The performance of studied models

Model	**Type**	**Average precision on COCO dev-set**	**Number of training stages**	**Precision on our dataset**	**Error On our dataset**	**Training time**
EfficientDet	One stage detector	55.1	3500 steps	52%	38%	12hours
YOLOv5	One stage detector	50.1	200 epochs	82%	30%	2hours
Detectron2	One stage detector	64.31	1500 steps	97%	42%	5 hours
Faster R-CNN	Two stage detector	36.8	10 000 steps	45%	80%	10 hours

The table shows the type of open-source model of our experiment as well as the number of epochs. The results obtained on the COCO datasets, as well as those results obtained on our dataset, which is expressed in terms of learning precision, and error, and finally of execution time. The results obtained improved as we increased the epoch number because the more we have more steps, the more we get more precision and little error. The dataset is also a determining element in object detection models. It is necessary to have a large dataset to achieve better results for most models. Also, obtaining good results requires using a GPU instead of a CPU [24]. Also, we are likely to get more performance when dropout is used.

The YOLOv5 model presented the best results found compared to the other models in terms of precision and error. The execution time was very reasonable compared to the number of epochs and the size of our database. However, YOLOv5 has medium results on the COCO dataset [25], but it proves that it is an efficient model on a medium dataset in a reduced time. So, we save these weights to use them in a robotic weed detection system based on Raspberry pi3. This will contribute to detecting weeds in real-time to achieve localized spraying of the weeds instead of propagating them on a large scale, which will be beneficial in terms of the quantity of herbicide exploited, thus protects the environment and saves time, this will be an automated decision-making without any human involvement [26].

4 Conclusion

We have presented in this paper a comparison approach based on the detection models of open-source objects, for this, we have used four recent models with different architectures and we have shown the different results obtained in terms of precision and error on our prepared dataset. The comparison of the results found showed that the number of epochs, the size of the dataset, Optimizing the GPU/ CPU assignment, data-augmentation are important factors for obtaining better results. The results obtained proved to us that YOLO v5 is a fast, lightweight model that we consider a better choice to be used in an on-board Raspberry based weed detection system that will be the subject of a future article.

5 References

[1] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," International journal of computer vision, vol. 128, no 2, p. 261-318, 2020. https://doi.org/10.1007/s11263-019-01247-4

[2] B. Jabir and N. Falih, "Digital agriculture in Morocco, opportunities and challenges," in 2020 IEEE 6th International Conference on Optimization and Applications (ICOA). IEEE, 2020. p. 1-5.https://doi.org/10.1109/icoa49421.2020. 9094450

[3] J. V. Stafford, "Implementing precision agriculture in the 21st century," Journal of agricultural engineering research, vol. 76, no 3, p. 267-275, 2000.https://doi.org/10. 1006/jaer.2000.0577

[4] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on knowledge and data engineering, vol. 22, no 10, p. 1345-1359, 2009.

[5] A. F. Khalifa, E. Badr, and H. N. Elmahdy, "A survey on human detection surveillance systems for raspberry pi," Image and Vision Computing, vol. 85, p. 1-13, 2019. https://doi.org/10.1016/j.imavis.2019.02.010

[6] J. Brownlee, “Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras,” Machine Learning Mastery, 2016.

[7] P. Goldsborough, "A tour of tensorflow," arXiv preprint arXiv:1610.01178, 2016.

[8] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 1251-1258. https://doi.org/10.1109/cvpr.2017.195
[9] J. Hao and T.K. Ho, "Machine learning made easy: A review of scikit-learn package in Python programming language," Journal of Educational and Behavioral Statistics, vol. 44, no 3, p. 348-361, 2019. https://doi.org/10.3102/1076998619832248
[10] S. Pinitkan and N. Wisitpongphan, "Abnormal Activity Detection and No-tification Platform for Real-Time Ad Hoc Network," International Journal of Online & Biomedical Engineering, vol. 16, no 15, 2020. https://doi.org/10.3991/ijoe.v16i15. 16065
[11] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020. p. 10781-10790. https://doi.org/10.1109/cvpr42600.2020.01079
[12] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in International Conference on Machine Learning. PMLR, 2019. p. 6105-6114.
[13] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," arXiv preprint arXiv:1506.01497, 2015.
[14] J. Pang, K. Chen, J. Shi, H. Feng, W. Ouyang, and D. Lin, "Libra r-cnn: Towards balanced learning for object detection," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019. p. 821-830. https://doi.org/10. 1109/cvpr.2019.00091
[15] V. Margapuri and M. Neilsen, "Seed Phenotyping on Neural Networks using Domain Randomization and Transfer Learning," arXiv preprint arXiv:2012.13259, 2020.
[16] [16] L. Boukhris, et al., "Tailored Deep Learning based Architecture for Smart Agriculture," in 2020 International Wireless Communications and Mobile Computing (IWCMC). IEEE, 2020. p. 964-969. https://doi.org/10.1109/iwcmc48107.2020.9148182
[17] Y. Saleh and G. issa, "Arabic Sign Language Recognition through Deep Neural Networks Fine-Tuning," International Journal of Online & Biomedi-cal Engineering, vol. 15, no 05, 2020. https://doi.org/10.3991/ijoe.v16i05.13087
[18] S. Abou El-Seoud, M. Siala, and G. McKee, "Detection and Classification of White Blood Cells Through Deep Learning Techniques," International Journal of Online & Biomedical Engineering, vol. 16, no 15, 2020.https://doi.org/10.3991/ijoe. v16i15.15481
[19] R. Yu, S. Zheng, A. Anandkumar, and Y. Yue, "Long-term forecasting using tensor-train rnns," Arxiv, 2017.
[20] H. J. Jeong, K. S. Park, and Y. G. Ha, "Image preprocessing for efficient training of YOLO deep learning networks," in 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE, 2018. p. 635-637.https://doi.org/10.1109/bigcomp. 2018.00113
[21] D. A. Van Dyk and X. L. Meng, "The art of data augmentation," Journal of Computational and Graphical Statistics, vol. 10, no 1, p. 1-50, 2001.
[22] W. Garcia-Quilachamin, J. S. Cano, L. P. Concepción, and E. S. Mantuano, "Validation of the Algorithm in the De-tection of the Image of a Person Based on the Control of a Lighting Device" International Journal of Online & Biomedical Engineering, vol. 16, no 09, 2020. https://doi.org/10.3991/ijoe.v16i09.12933
[23] Z. Chen, F. Yang, A. Lindner, G. Barrenetxea, and M. Vetterli, "Howis the weather: Automatic inference from images," in 2012 19th IEEE International Conference on Image Processing. IEEE, 2012. p. 1853-1856. https://doi.org/10.1109/icip.2012.6467244
[24] Y. E. Wang, "Benchmarking tpu, gpu, and cpu platforms for deep learning," arXiv preprint arXiv:1907.10701, 2019.
[25] Y. Li, K. Yin, J. Liang, C. Wang, and G. Yin, "A Multi-task Joint Framework for Real-time Person Search," arXiv preprint arXiv:2012.06418, 2020.
[26] B. Jabir, N. Falih, and K. Rahmani, "HR analytics a roadmap for decision making: case study," Indonesian Journal of Electrical Engineering and Computer Science, 2019, vol. 15, no 2, p. 979-990. https://doi.org/10.11591/ijeecs.v15.i2.pp979-990

6 Authors

Brahim Jabir received his Master degree in 2015 in computer engineering and systems at the LIMATI Laboratory of Polydisciplinary Faculty, Sultan Moulay Slimane University in Beni Mellal, Morocco. Currently, He is a Ph.D. student in the Faculty of Sciences and Technics of the same University and he is working as a teacher of computer science in the regional centers of education and training professions in Beni Mellal, Morocco. His research interest is about Digital Agriculture, Deep learning, Strategic Analytics and Information Systems. (ibra.jabir@gmail.com)

Noureddine Falih is PhD on Computer Science from Faculty of Sciences and Technologies of Mohammedia, Morocco in 2013. He is an associate professor in LIMATI Laboratory of Polydisciplinary Faculty, Sultan Moulay Slimane University at Beni Mellal, Morocco since 2014. He has 18 years of professional experience in several renowned companies. His research topics are about Information System Governance, Business Intelligence, Big Data Analytics and Digital Agriculture. (nourfald@yahoo.fr)

Khalid Rahmani received his Ph.D. in physics "Study of the optoelectronic properties of a donor confined in an inhomogeneous quantum well and quantum dots" in 2012, currently He is a Teacher and researcher in computer science at LIMATI Laboratory of Polydisciplinary Faculty, Sultan Moulay Slimane University, His research interest is: Quantum Physics, Materials Science, Solid State Physics. (kh.rahmani@hotmail.fr)

Article submitted 2021-02-07. Resubmitted 2021-03-30. Final acceptance 2021-04-01. Final version published as submitted by the authors.