Computational Comparison of Deep Learning Algorithms for Object Detection

Vasileios Balafas*
Nikolaos Ploskas*
v.balafas@uowm.gr
nploskas@uowm.gr
Department of Electrical and Computer Engineering, University of Western Macedonia
Kozani, Greece

ABSTRACT

Finding and recognizing different objects in an image quickly and reliably is an important field of computer vision. Object detection is a challenging problem in this field. Humans have the ability to perform such complex tasks fast and accurately. In contrast, the problem of locating objects via a computer is not so simple. Deep learning algorithms have emerged as powerful methods to detect objects in an image. In this paper, six state-of-the-art object detection algorithms are presented, analysed and compared computationally using four different datasets, two single class and two multiple class datasets. The computational results show that the algorithms achieve higher accuracy on the single class datasets than the multi class datasets.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Machine learning algorithms**; **Neural networks**.

KEYWORDS

Object detection, Computer vision, Machine learning, Neural networks

ACM Reference Format:
Vasileios Balafas and Nikolaos Ploskas. 2021. Computational Comparison of Deep Learning Algorithms for Object Detection. In *25th Pan-Hellenic Conference on Informatics, 26 - 28 November 2021, Volos, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3503823.3503838

1 INTRODUCTION

Amongst the most fundamental challenges in computer vision is object detection. Object detection is a computer vision task that detects instances of objects of specific classes (such as persons, animals, cars, or buildings) in digital pictures like photos or video frames. It is the foundation of many other computer vision tasks such as image captioning, object tracking, segmentation, and others. Some examples of object detection applications include face detection, number-plate identification, people counting, and text detection. The purpose of object detection is to generate computational models that satisfy the most basic question that computer vision applications possess: "What objects are there and where are they?".

Object detection is often confused with image recognition. The most important difference between them is that image recognition only assigns a label to an image. More precisely, a photo of a car is labeled "car". A photo of two cars is still labeled "car". On the other hand, object detection draws a bounding box around each car and assigned to the box a label with the "car" tag. The model tries to identify all objects in an image that has been trained to detect and which label should be applied to each object. Thus, object detection provides more information about an image than recognition.

Today, the problem of recognizing and locating objects in images is solved using Convolutional Neural Networks (CNNs). With the introduction of CNNs and the adaption of computer vision technologies, object detection became much more common in the current generation. The fact that CNNs, combined with graphic cards, make it possible to solve object detection and tracking problems in less time, enables them to be used in real-time applications. In this regard, deep learning led to a significant improvement in object detection. The problem of object detection has been studied a lot with with the first publications appearing in the 90s [21] although comparisons between different object detection algorithms are still missing. This work aims to fill this gap. This paper aims to make a thorough analysis of the most prevalent deep learning object detection algorithms and compare them in real-world problems.

The paper is organized as follows. Section 2 includes a brief introduction to CNNs and presents the object detection algorithms that we study in this computational comparison. In Section 3, we review the related work in this field. Section 4 presents the insights that we have gathered from our computational study of the object detection algorithms. Finally, conclusions are provided in Section 5.

2 OBJECT DETECTION ALGORITHMS

2.1 Convolutional neural networks

CNNs are algorithmic machine learning algorithms that take an input image, assign meaning (weights and biases) to different objects of the image, and distinguish between them. The preprocessing required in a CNN is much less compared to other categorization algorithms. While primitive methods create the filters by hand,

*Both authors contributed equally to this research.

PCI 2021, 26 - 28 November 2021, Volos, Greece

ACM ISBN 978-1-4503-9555-7/21/11.
https://doi.org/10.1145/3503823.3503838

CNNs can learn these filters/features with enough training. The architecture of a CNN is analogous to the connectivity of neurons in the human brain and thus, it was inspired by the organization of the visual brain. In neural networks, CNNs are one of the main categories for image recognition, image categorization, face recognition, and of course object detection. CNNs process images using weighting tables called filters (sometimes referred to as features) that recognize certain features, such as vertical edges, horizontal edges, etc. The ultimate goal of CNNs is to find out what is going on in the scene.

For this study, we decided to study and compare six state-of-the-art object detection algorithms: Faster R-CNN, SSD, RetinaNet, EfficientDet, YOLOv3, and YOLOv4. Faster R-CNN belongs to the family of region proposal algorithms and the remaining algorithms are single-stage detectors. The main difference between region proposal algorithms and single-stage detectors is that the region proposal algorithms use a CNN to extract the features from a given image and then propose regions where an object might be located, and finally pass this region to the object classifier. A single-stage detector, on the other hand, performs full object detection in the same CNN. In the next subsections, we briefly introduce these algorithms.

2.2 Faster R-CNN

Faster R-CNN [16] belongs to the family of R-CNNs. These algorithms use selective search to find area suggestions. Selective search is a time-consuming and sluggish operation that has an impact on the algorithm's performance. Therefore, Ren et al. [16] proposed Faster R-CNN, which is an object detection algorithm that eliminates the need for the selective search algorithm and allows the network to learn area suggestions. The image is used as input to a CNN, which provides a map of convolutional features. Instead of using a selective search algorithm on the feature map to propose suggestions, a separate network is used to predict region suggestions. The predicted region proposals are then reconfigured using a region of interest layer, which is then used to rank the image within the proposed regions and predict bounding box values.

The Faster R-CNN algorithm requires multiple passes on each image to export all objects. The overall mAP@.5 on the Common Objects in Context (COCO) dataset [12] was 42.7% and the mAP@[.5,.95] was 21.9%, as shown in [16].

2.3 Single Shot Detector (SSD)

The SSD algorithm was proposed by Liu et al. [13] in 2016 and became a state-of-the-art detector. The mAP@.5 on the COCO dataset was 46.5% at 59 frames per second, and in Pascal VOC12, the mAP@.5 was 74.9%. Unlike the Faster R-CNN, which uses a subnet to suggest regions, the SSD detectors rely on a set of predefined regions. A grid of anchor points is placed over the input image, and at each anchor point, frames of different shapes and sizes serve as regions. For each frame at each anchor point, the model predicts whether or not an object is within the area, and changes the position and size of the frame to fit closer to the object. Since there are many frames at each anchor point and the anchor points may be close to each other, SSD detectors generate many potential overlaps. Finally, post-processing of the SSD output is performed to remove most of these predictions and select the best one.

The architecture of SSD is based on VGG-16 architecture but without the fully connected layers. Instead of the original fully connected VGG layers, a number of cohesive auxiliary layers (starting from conv6) have been added to allow export of features at multiple scales and gradually reduce the input size at each subsequent layer. The backbone of SSD is MobileNet v2 [17], which is based on a reverse residual structure where the input and output of the residual block are thin bottleneck layers. On the other hand, typical residual models use extended representations in the input. MobileNet v2 uses light convolutions in depth to filter features in the middle expansion layer.

2.4 RetinaNet

RetinaNet [11] is one of the best single-level object detection models, proven to work well with dense and small objects. RetinaNet has been configured to have two improvements over existing single-stage object detection models: Feature Pyramid Networks (FPN) [10] and Focal Loss (FL) [11]. The main problem is the strong imbalance between foreground and background classes that occurs when training the dense detector. To address this, RetinaNet introduced a new loss function called FL; this loss function transforms the conventional cross-entropy loss to allow the detector to focus more on hard misclassified cases during training. With FL, single-stage detectors can achieve the same accuracy as two-stage detectors while maintaining a fast detection speed. RetinaNet also uses the FPN proposed as the backbone of the network, which in turn is built on ResNet [8] and is fully contiguous. The fully convergent nature allows the network to take an image of any size and export feature maps of similar size to multiple layers in the feature pyramid. The FPN design includes two paths connected by side connections. RetinaNet's mAP@0.5 on the COCO dataset was 59.1% and mAP@[.50,0.55,..,0.9] was 39.1%.

2.5 EfficientDet

EfficientDet [20] is a set of eight algorithms from Google that are very efficient and accurate. It can perform well on a variety of devices and is always comparable with a variety of resource constraints. In particular, in the case of one model and one scale, the EfficientDet-D7 achieved 52.2% mAP in the overall COCO dataset with 52M parameters and 325B FLOPs. Compared to existing algorithms, the number of parameters was reduced by four to nine times and the FLOPs by 13 to 42 times.

To develop this algorithm, Google first proposed a weighted two-way feature pyramid network (BiFPN) that offers easy and fast merging at multiple scales, and second, a scaling method for complex pyramids that scales the depth and width, feature grid, and box/class prediction grid of all backbone networks. EfficientDet detectors are single-shot detectors such as SSD and RetinaNet. The backbone of the EfficientDet network is the EfficientNet trained in the ImageNet [7] dataset.

2.6 YOLO V3

All previous object detection algorithms use regions for locating the objects in an image. The network does not see the whole image, but only some parts of the image that are more likely to contain an object. YOLO is short for "You Only Look Once" [14]; it is an object detection algorithm that is very different from region-based algorithms. Since the entire recognition process runs on a single CNN, the recognition performance can be optimized end-to-end. A fast version of YOLO runs at 155fps with mAP@0.5 of 52.7% on the VOC07 dataset, while its improved version runs at 45fps with mAP@0.5 of 63.4% on the same dataset.

The YOLOv3 algorithm [15] first divides an image into an SxS grid. Each grid cell predicts a certain number of bounding boxes around objects that score high in the predefined classes. Each bounding box has a corresponding confidence value indicating how accurate the prediction should be, and only one object per boundary box is detected. The boundary boxes are generated by clustering the dimensions of the ground truth boxes from the original dataset to find the most common shapes and sizes. YOLO is trained to perform classification and bounding box regression simultaneously.

2.7 YOLO V4

YOLOv4 [6] is a significant improvement over YOLOv3. The implementation of the new architecture in Backbone and the changes in Neck have achieved excellent results with real-time crawl speed. In the MS COCO dataset, YOLOv4 achieves an average accuracy of 43.5 % AP and a speed of 65 FPS on a Tesla V100 GPU. To achieve these results, it combines features such as weighted connections, Cross-Stage-Partial connections (CSP), Self-Adversial training (SAT), Mish activations, data increments, DropBlock normalization, and CIoU deviation.

The backbone architecture of YoloV4 consists of three parts: (a) Bag of freebies, (b) Bag of specials, and (c) CSPDarknet53. Bag of freebies methods are a set of methods that only increase the training cost or change the training strategy while keeping the inference cost low. Some of these methods are data augmentation, photometric bias, geometric bias, MixUp, and CutMix. Bag of specials is a set of methods that slightly increase the cost of the detection, but significantly improve the accuracy of the detector. These methods include the introduction of attention mechanisms that expand the receptive field of the model and improve the ability to integrate features. CSPDarknet53 is the backbone network for feature extraction of YOLOv4.

3 RELATED WORK

Srivastava et al. [19] compared the Faster R-CNN, SSD, and YOLOv3 algorithms. Using the COCO dataset, they evaluate the accuracy, precision, and F1 score of the algorithms. From the results of the study, they conclude that the superiority of one of the algorithms over the other two is highly dependent on the use cases in which they are used. In an identical test environment, YOLOv3 performs better than SSD and Faster R-CNN, making it the best of the three algorithms.

John et al. [9] compared the algorithms R-CNN, Fast R-CNN, Faster R-CNN, SSD, and YOLOv3 in the Pascal VOC and COCO datasets. The performance and accuracy are the main criteria in their comparative analysis. They conclude that with the development of the model, the speed and accuracy were improved and increased. Fast R-CNN is better than R-CNN, but Faster R-CNN is much better than Fast R-CNN. Moreover, SSD is better than Faster R-CNN, while YOLO v3 is better than SSD. They conclude that YOLOv3 is extremely fast and accurate.

Since most related work studies compare object detection algorithms using the two large datasets, Pascal VOC and Microsoft COCO, we decided to study the performance and accuracy of the algorithms by fine-tuning the object detection models in other datasets, some of which are very small, with only one class, and others larger, with 7 and 30 classes. Thus, in Section 4, we present our computational analysis of six state-of-the-art algorithms on four different datasets.

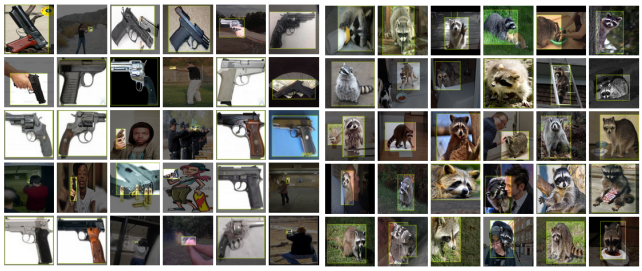
4 COMPUTATIONAL STUDY

In this Section, we train and evaluate the performance and the accuracy of the algorithms: Faster R-CNN, RetinaNet, SSD, EfficientDet, YOLOv3, and YOLOv4 in four different datasets. Two of our datasets consist of a single class and the other two datasets contain multiple classes. The computational comparison has been performed on an Intel Core i7-7820X 3.60GHz processor with 32 GB of main memory and 16 cores, a clock of 3,700 MHz, an L1 data cache of 11 MB per core and a memory bandwidth of 85 GB/s, running under Ubuntu 20.10 64-bit, and on a GeForce GTX 1080 Ti GPU with 11 GB GDDR5 352-bit memory, a core clock of 1683 MHz, a memory clock of 11,124 MHz and a memory bandwidth of 484 GB/s.

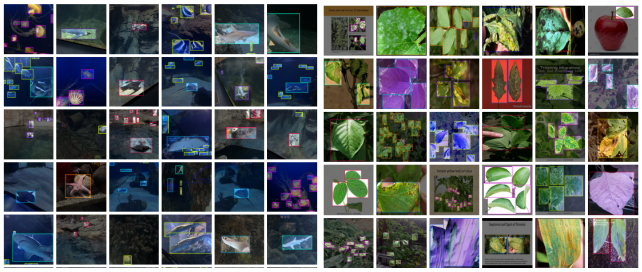
The first dataset is the Pistols Dataset from the University of Grenada [4]. It contains 2,986 images and 3,448 tags in a single object category: pistol. The images are wide-ranging: handguns, cartoons, and studio-quality gun images. The second single class dataset is the Raccoon dataset [2] created by Dat Tran. This dataset contains 196 images of raccoons and 213 bounding boxes as some images have two raccoons in an image. This is a single class problem and images vary in dimensions. The third dataset is the Aquarium Dataset [5] and contains seven different classes of objects. This dataset consists of 638 images collected by Roboflow from two aquariums in the United States. The final dataset is the PlantDoc dataset [18]. The PlantDoc dataset consists of approximately 2,600 images and 30 classes (diseased and healthy) of 13 different plant species.

Figure 1 shows the visualization of the four datasets (images along with the frames around the objects we are interested in) used for training the algorithms of the study.

Moreover, the TensorFlow 2 Object Detection API and the Darknet framework were used to train and evaluate the object detection models. The TensorFlow 2 Object Detection API [3] is the framework that used to train and evaluate the algorithms Faster R-CNN, SSD, RetinaNet and EfficientDet alogn with pre-trained models for every algorithm in the COCO dataset. On the other hand, Darknet [1] is the official framework for the YOLO algorithms. For the training process, we used the following hyperparameters. For the Faster R-CNN the learning rate that we set was 0.04, the total steps were 2,000, the warm-up learning rate was 0.013333, and warm-up steps were 2,000. For the SSD algorithm, the learning



(a) Pistols dataset (b) Raccoon dataset



(c) Aquarium dataset (d) Plant Doc dataset

Figure 1: Visualization of the four datasets

Algorithm	Backbone	Framework
Faster R-CNN	Resnet50	TF 2 Object Detection API
SSD	MobileNet v2	TF 2 Object Detection API
RetinaNet	Resnet50	TF 2 Object Detection API
EfficientDet	EfficientNet	TF 2 Object Detection API
YOLOv3	Darknet-53	Darknet
YOLOv4	CSPDarknet53	Darknet

Table 1: The framework and the backbone network for each object detection algorithm.

rate was 0.800000011920929, total steps were 2, 000, warm-up learning rate was 0.13333000242710114, and warm-up steps were 2, 000. RetinaNet learning rate base was 0.039999910593033 total steps were 2, 000, warm-up learning rate was 0.013333000242710114, and warm-up steps were 2, 000. For EfficientDet learning rate base was 0.079999821186066 total steps were 300000, warm-up learning rate was 0.0010000000474974513, and warm-up steps were 2, 500. For YOLOv3 learning rate was 0.001, burn in was 1, 000, max batches were 2, 000, and steps were 1600 and 1800.0. For YOLOv4 learning rate was 0.001, burn-in was 1, 000, max batches were 2, 000, steps were 1, 600 and 1800.

Table 1 shows the framework and backbone network that have been used for training each object detection algorithm.

Furthermore, for the evaluation of the total accuracy of the object detector models, we used some of the metrics of the COCO challenge. More precisely, the metrics that we utilized are the following: (i) AP: which is mAP averaged over ten Intersection over Union (IoU) thresholds (i.e., 0.50, 0.55, 0.60, . . . , 0.95) and is the main metric of the COCO challenge, (ii) $AP^{\text{IoU=.50}}$: AP at IoU=0.50, and (iii) $AP^{\text{IoU=.75}}$: AP at IoU=0.75.

Dataset	Pistols Dataset			Raccoon Dataset		
Algorithm	**AP**	**AP50**	**AP75**	**AP**	**AP50**	**AP75**
Faster R-CNN	0.638	0.884	0.682	0.739	0.999	0.957
SSD	0.547	0.789	0.597	0.608	0.954	0.685
EfficientDet	0.572	0.855	0.617	0.669	0.972	0.751
RetinaNet	0.535	0.762	0.564	0.577	0.949	0.590
YOLOv3	0.593	0.893	0.655	0.564	0.985	0.614
YOLOv4	0.604	0.940	0.709	0.584	0.957	0.706

Table 2: Accuracy of the algorithms according to the COCO challenge metrics on the single class datasets.

Dataset	Aquarium Dataset			PlantDoc Dataset		
Algorithm	**AP**	**AP50**	**AP75**	**AP**	**AP50**	**AP75**
Faster R-CNN	0.350	0.662	0.330	0.441	0.607	0.535
SSD	0.349	0.653	0.311	0.238	0.329	0.284
EfficientDet	0.354	0.657	0.333	0.235	0.352	0.287
RetinaNet	0.378	0.647	0.374	0.278	0.382	0.328
YOLOv3	0.444	0.756	0.586	0.421	0.570	0.471
YOLOv4	0.339	0.686	0.300	0,151	0.258	0.153

Table 3: Accuracy of the algorithms according to the COCO challenge metrics on the multi class datasets.

We trained each algorithm in the study on each dataset and evaluated the mAP of each trained model. Tables 2 and 3 show the evaluated results of the trained models of the algorithms Faster R-CNN, SSD, EfficientDet, RetinaNet, YOLOv3, and YOLOv4 according to the measurements used to characterize the performance of an object detector in the COCO challenge. Table 2 shows the evaluation results of the trained models on single class datasets, while Table 3 shows the evaluation results of the trained models on multiclass datasets.

An important parameter in object detection is the inference time of the trained models for both CPUs and GPUs. Therefore, we decided to measure the inference times of the trained models. In Table 4, we present the average inference time for each object detection algorithm to detect objects in an image. In this study, inference was performed exclusively using either the CPU, in our case an i7-7820X CPU @ 3.60GHz, or by acceleration using the GPU, in our case the Nvidia GeForce GTX 1080 Ti. It can be clearly seen that all object detection algorithms in this study have significantly higher performance when run on a GPU and not only on a CPU. The only exception was the SSD algorithm using MobileNetV2, a very efficient mobile backbone optimized for performance.

It is very difficult to make a fair comparative analysis between different object detectors. There is no simple answer to the question of which model is the top. For each different application, we have to make our choice to find the best compromise between accuracy and speed. In addition to detector types and algorithms, other options also affect performance, such as feature exporters (VGG16, ResNet, Inception, MobileNet), input image resolution, training dataset, loss

Algorithm	GTX 1080 Ti GPU	i7-7820X CPU
Faster R-CNN	0.29 sec.	0.73 sec.
SSD	0.05 sec.	0.08 sec.
EfficientDet	0.07 sec.	0.16 sec.
RetinaNet	0.15 sec.	0.33 sec.
YOLOv3	0.11 sec.	6.11 sec.
YOLOv4	0.16 sec.	6.93 sec.

Table 4: Object detection times in an image on CPU and GPU.

function, deep learning software, and more. From the results, we can see that the algorithms have higher accuracy on single-class datasets than on multi-class datasets.

5 CONCLUSIONS

In this paper, we studied the object detection problem using machine learning algorithms. We trained six state-of-the-art object detection algorithms on four different datasets and compared their accuracy and performance. As for the algorithms, R-CNNs are accurate but quite slow, even when running on GPUs. In our experiments, we came to the same conclusion. The Faster R-CNN algorithm had average accuracy, but its GPU execution time was about 0.29 seconds. For single-stage detectors, the YOLOv4 and EfficientDet algorithms also had high accuracy, but their average detection time was lower.

REFERENCES

[1] [n. d.]. *AlexeyAB/darknet: YOLOv4v / Scaled-YOLOv4 - Neural Networks for Object Detection (Windows and Linux version of Darknet).* https://github.com/AlexeyAB/darknet

[2] [n. d.]. *Github: Raccoon Dataset.* https://github.com/datitran/raccoon_dataset

[3] [n. d.]. *TensorFlow Object Detection API - Github.* https://github.com/tensorflow/models/tree/master/research/object_detection

[4] [n. d.]. *Weapons detection for security and video surveillance | Soft Computing and Intelligent Information Systems.* https://sci2s.ugr.es/weapons-detection

[5] 2021. Aquarium Object Detection Dataset. https://public.roboflow.com/object-detection/aquarium

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934 [cs.CV]

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Anand John and Divyakant Meva. 2020. A Comparative Study of Various Object Detection Algorithms and Performance Analysis. October (2020). https://doi.org/10.26438/ijcse/v8i10.158163

[10] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2016. Feature Pyramid Networks for Object Detection. arXiv:1612.03144 [cs.CV]

[11] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. arXiv:1708.02002 [cs.CV]

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2015. Microsoft COCO: Common Objects in Context. arXiv:1405.0312 [cs.CV]

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science* (2016), 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640 [cs.CV]

[15] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *arXiv* (2018).

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs.CV]

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[18] Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat, and Nipun Batra. 2019. PlantDoc: A Dataset for Visual Plant Disease Detection. arXiv:1911.10317 [cs.CV]

[19] Shrey Srivastava, Vishvas Divekar, Chandu Anilkumar, Ishika Naik, Ved Kulkarni, and V Pattabiraman. [n. d.]. Comparative analysis of deep learning image detection algorithms. ([n. d.]). https://doi.org/10.1186/s40537-021-00434-w

[20] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. arXiv:1911.09070 [cs.CV]

[21] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. 2019. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055* (2019).