Computers and Electronics in Agriculture 175 (2020) 105535

Contents lists available at ScienceDirect

Computers and Electronics in Agriculture

journal homepage: www.elsevier.com/locate/compag

Original papers

Vineyard trunk detection using deep learning – An experimental device benchmark

André Silva Pinto de Aguiar[a,b,*], Filipe Baptista Neves dos Santos[a], Luís Carlos Feliz dos Santos[a,b], Vitor Manuel de Jesus Filipe[a,b], Armando Jorge Miranda de Sousa[a,c]

[a] INESC TEC – INESC Technology and Science, 4200-465 Porto, Portugal
[b] School of Science and Technology, University of Trás-os-Montes e Alto Douro, 5000-801 Vila Real, Portugal
[c] Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal

ARTICLE INFO

Keywords:
Deep learning
Object detection
Agricultural robots

ABSTRACT

Research and development in mobile robotics are continuously growing. The ability of a human-made machine to navigate safely in a given environment is a challenging task. In agricultural environments, robot navigation can achieve high levels of complexity due to the harsh conditions that they present. Thus, the presence of a reliable map where the robot can localize itself is crucial, and feature extraction becomes a vital step of the navigation process. In this work, the feature extraction issue in the vineyard context is solved using Deep Learning to detect high-level features – the vine trunks. An experimental performance benchmark between two devices is performed: NVIDIA's Jetson Nano and Google's USB Accelerator. Several models were retrained and deployed on both devices, using a Transfer Learning approach. Specifically, MobileNets, Inception, and lite version of You Only Look Once are used to detect vine trunks in real-time. The models were retrained in a built in–house dataset, that is publicly available. The training dataset contains approximately 1600 annotated vine trunks in 336 different images. Results show that NVIDIA's Jetson Nano provides compatibility with a wider variety of Deep Learning architectures, while Google's USB Accelerator is limited to a unique family of architectures to perform object detection. On the other hand, the Google device showed an overall Average precision higher than Jetson Nano, with a better runtime performance. The best result obtained in this work was an average precision of 52.98% with a runtime performance of 23.14 ms per image, for MobileNet-V2. Recent experiments showed that the detectors are suitable for the use in the Localization and Mapping context.

1. Introduction

The Oporto vineyards, Fig. 1, are located in the Douro Demarched Region, the oldest controlled winemaking region in the world, a UNESCO heritage place (Andresen et al., 2004). These vineyards are built in steep slope hills, which brings several challenges to the development of robotic solutions in this context. The characteristics of the hill cause signal blockage that decreases the accuracy of signals emitted by the Global Navigation Satellite System (GNSS), making unreliable the use of, for example, the Global Positioning System (GPS). Also, the terrain highly characterized by irregularities leads to high inaccuracy of sensors like wheel odometry and Inertial Measurement Units (IMU)s (Dos Santos et al., 2016).

The vast extension of the vineyard and the challenging conditions that they present lead to an increasing need for the substitution of human labor by automatic and autonomous machines. These machines can be used to perform operations such as planting, harvesting, environmental monitoring, supply of water and nutrients (Roldán et al., 2018) and have the potential to transform and have a significant impact in many agricultural economic sectors (Duckett et al., 2018). For mobile robots, the capability of autonomously navigating in steep slope vineyards has a mandatory requirement: real-time localization. In order for a robot to navigate safely in the vineyard, it needs to be able to localize itself. Feature-based localization is one of the most common approaches to do so (Santos et al., 2015; Cheein et al., 2011; Aguiar et al., 2019). However, the extraction of reliable and persistent features in an outdoor environment is a challenging task. In the vineyard context, it makes sense to provide the robot with the ability to recognize vine trunks as high-level features to use in the localization and mapping process. The robot can be endowed with camera systems and artificial

* Corresponding author.
E-mail addresses: andre.s.aguiar@inesctec.pt (A.S. Pinto de Aguiar), fbsantos@inectec.pt (F.B. Neves dos Santos), luis.c.santos@inesctec.pt (L.C. Feliz dos Santos), vfilipe@utad.pt (V.M. de Jesus Filipe), asousa@fe.up.pt (A.J. Miranda de Sousa).

https://doi.org/10.1016/j.compag.2020.105535
Received 3 March 2020; Received in revised form 25 May 2020; Accepted 25 May 2020
Available online 20 June 2020

Fig. 1. Typical steep slope vineyard in the Douro's region.

intelligence to learn what a trunk is. This image recognition capability can also be used for management and supervision purposes when combined with data analysis (Jin et al., 0000; Bai et al., 2019). To perform such tasks, Deep Learning (DL)-based object detection (Zhao et al., 2019) can be used. DL (LeCun et al., 2015; Goodfellow et al., 2016) allows a machine to learn to classify, detect, and segment objects using a given training dataset. Convolutional Neural Networks (CNN) are widely used to perform such a task. They showed the highest levels of performance in several contests in machine learning and pattern recognition (Schmidhuber, 2015). Despite this, training a CNN from scratch, and obtaining accurate results while deploying it on a real scenario, assumes that both training and test data must be in the same feature space, and have the same distribution (Pan and Yang, 2010). However, in some real-world scenarios, data collection can be challenging and time-expensive. To overcome this limitation, learners can be trained with data easily collected from different domains (Weiss et al., 2016; Lu et al., 2015; Shao et al., 2015). In other words, the learning procedure can be performed transferring knowledge from a given task that was already learned, and the training procedure can focus on a subset of layers of the CNN. This methodology is called Transfer Learning (TL) (Torrey and Shavlik, 0000).

In this work, the feature extraction problem in the vineyard context is addressed using TL to detect vine trunks. To do so, CNNs models based in the Single Shot Multibox (SSD) (Liu et al., 0000), Pooling Pyramid Network (PPN) (Jin et al., 2018), and SSDLite (Sandler et al., 2018) architectures are considered, such as several versions of the MobileNets (Howard et al., 2017), with slighly variations on the hyperparameters. Also, a version of Inception (Szegedy et al., 2015) built on top of SSD was considered. Finally, a lite version of YOLO - Tiny YOLO-V3 (Redmon and Farhadi, 2018; Joseph Chet Redmon et al., 2020) – was trained and tested. Two low-cost devices are used to perform real-time inference: Google's USB Accelerator (Google et al., 2020) and NVIDIA's Jetson Nano (Corporation et al., 2020). An experimental benchmark between the two devices is performed, using a built in–house dataset that is publicly available (http://vcriis01.inesctec.pt – `DS_AG_39`). The devices are compared considering the Average Precision (AP) of trunk detection resultant from the deployment of the models in each one, and the respective inference time.

The rest of the paper is described as follows. In the state-of-the-art, the related work is reviewed. Section 3 provides some basic information about the DL-based concepts and tools used in this work. Section 4 describes the two devices used in this work to perform trunk detection. Section 5 contains the methodology adopted, such as the data collection method, the training procedure, and the inference approaches. Section 6 presents the proposed system results using the built in–house dataset, and the respective analysis and discussion. Finally, the work is summarized in Section 7.

Table 1
Tensorflow-based trained models.

Model	Version	Architecture	α	Resolution
MobileNet	1	SSD	1	300 × 300
MobileNet	1	SSD	0.75	300 × 300
MobileNet	1	PPN	1	640 × 640
MobileNet	2	SSD	1	300 × 300
MobileNet	2	SSDLite	1	300 × 300
Inception	2	SSD	1	300 × 300

2. Related work

Image classification and object detection based on DL techniques are widely present in the agriculture sector. Intensive and time expensive tasks are being replaced by automatic machines, endowed with artificial intelligence. These machines are performing operations in the agriculture context such as plant disease detection, weed identification, seed identification, fruit detection and counting, obstacle detection, and others (Santos et al., 2020; Kamilaris and Prenafeta-Boldú, 2018; Kamilaris and Prenafeta-Boldú, 2018). Table 1 provides a summary of DL-based object detection works in the agricultural field.

From Table 1, several conclusions can be extracted. The majority of works focus on fruit detection, mainly in orchards. Relatively to these works, the most common application is fruit counting. Additionally, a minority of the state-of-the-art focuses on insect detection for pest identification, and, also, obstacle detection. Overall, the majority of works present high performance. Even with significantly different dataset sizes, in general, the works achieve AP or F1 scores higher than 80%.

Dias et al. (2018) fine-tune a pre-trained CNN to detect apple flowers. This work uses data augmentation to quadruple the original dataset size. To evaluate the proposed DL-based detector, both Precision vs Recall (PR) and the F1 score were used. This way, both the optimal performance of the method through F1 score, and the expected performance across a range of decision thresholds through Average Precision (AP) are evaluated. This work achieves a F1 score of 92.1% and a AP of 97.2%. CropDeep (Zheng et al., 000) proposes the largest dataset from all the analysed works, with 31,147 images of crop species with a total of 31 different classes. The authors train and test several state-of-the-art models for object detection. For example, for ResNet (He et al., 2016) they achieved a medium AP of 92.79%. In Koirala et al. (2019) several state-of-the-art DL architectures are trained to detect mango fruits in orchards. With the same purpose, the authors propose a new architecture based on YOLO, called MangoYOLO. From this architecture, two models were created, differing on the weights initialization. The first, MangoYOLO(s), was trained from scratch, while the second, MangoYOLO(pt), was pre-trained on the COCO dataset (Lin et al., 2014). The training set is composed of 11,820 fruits. MangoYOLO (s) achieves an F1 score of 96.7% and an AP of 98.6%, while MangoYOLO(pt) results in an F1 score of 96.8% and an AP of 98.3%. To detect apples during different growth stages in orchards, Tian et al. (2019) proposed an improvement to the state-of-the-art model YOLO-V3 (Redmon and Farhadi, 2018). The dataset used is composed of high-resolution images of apples in three different growth stages. The method achieved an F1 score of 81.7% for the training set containing three types of growth stages. Similarly, Bargoti and Underwood (2017) proposes the use of state-of-the-art architecture Faster R-CNN (Ren et al., 2015) for fruit detection in orchards, including mangoes, almonds, and apples. In the first stage, the architecture was used without any modification, using TL. In the second stage, in order to improve fruit detection in images with a larger number of fruits, a modification

was proposed to the raw architecture. Using all the training data, the authors obtained an F1 score of 90.4% for apples, 90.8% for mangoes and 77.5% for almonds. DeepFruits (Sa et al., 0000) proposes a DL-based fruit detector fine-tuning the state architecture Faster-RCNN. The VGG-16 (Simonyan and Zisserman, 2014) model, previously trained using ImageNet (Deng et al., 2009), is adapted through the use of a dataset with RGB and Near-Infrared (NIR) images and three classes named background, sweet pepper, and rock melon. As the best result, the authors obtained an F1 score of 83.8%. L*a*b*Fruits (Kirk, 0000) combines a visual processing approach with one-stage DL networks to detect strawberries. The entire dataset is constituted by 890 ripe strawberries and 3329 unripe strawberries. While testing, the authors consider that a detection is correct when the Interception over Union (IoU) is at least 0.5. Results show that this work obtained an F1 score of 74.4% for RGB images considering both classes.

To detect and count pest in images, Ding and Taylor (2016) is proposed. To test the proposed pipeline, the authors use the Interception-Over-Minimum (IOMin) concept with a threshold level of 0.5. Two different evaluation metrics were used: miss rate vs false positives per image (FPPI) and AP, in order to focus on the trade-off between reducing miss detections and reducing false positives. The best result achieved from this work is 93.1% for AP and 9.9% for miss rate vs FPPI. Also in this context, Zhong et al. (2018) detect, count and classify 6 types of flying insects for pest control in the agricultural context. The detection pipeline uses a single class and is built using state-of-the-art YOLO architecture pre-trained on ImageNet. The evaluation is performed using the counting accuracy, defined as the ratio of a correctly detected number to the total number of detected flying insects. The final counting accuracy obtained by YOLO was 93.71%.

For obstacle detection, Steen et al. (0000) proposes the fine-tune of a state-of-the-art CNN for detection of a specific object. The detector was evaluated in row crops context, obtaining a precision of 99.9% and recall of 36.7%, and in mowing grass, obtaining a precision of 90.8% and a recall of 28.1%.

3. Deep learning background

This work uses two sets of models based on the SSD architecture (Liu et al., 0000) to detect vine trunks, the MobileNets (Howard et al., 2017), and Inception-V2 (Szegedy et al., 2015). Tiny YOLO-V3 (Redmon and Farhadi, 2018; Joseph Chet Redmon et al., 2020) is also used to perform this task. This section presents an overview of the SSD-based architectures and the all the type of models used.

3.1. Single shot multibox

SSD, Fig. 2, is based on a feed-forward CNN that detects objects producing a fixed number of bounding boxes and scores. This architecture is build upon a NN based on a given standard architecture. Its main modules are:

- Convolutional feature layers that decrease progressively in size, detecting objects at multiple scales.
- Convolutional filters represented on top of Fig. 2, that produce a fixed number of detection predictions.
- A set of bounding boxes associated with each feature map cell.

These characteristics allow to detect objects at multiple scales, i.e., objects of different sizes in images with different resolutions.

3.2. Single shot multibox lite

SSDLite was proposed by Sandler et al. together with MobileNet-V2 (Sandler et al., 2018), being based on the design of MobileNets. This architecture, as the name suggests, is a lighter version of the original SSD. The conventional convolutions present in the SSD architecture are replaced by separable convolutions. These ones split full convolutional operations into two separable layers. The first, called depthwise convolution, applies a single convolution to each image channel. The second, a 1×1 convolution called pointwise convolution, builds new features computing linear combinations of the input image channels. This design revealed to be much more computationally efficient, highly reducing the CNN size. These characteristics are directly oriented to mobile and embedded devices.

3.3. Pooling pyramid network

PPN (Jin et al., 2018), similarly to SSDLite, is a reduced size version of SSD. The architecture was designed in order to run faster than SSD, maintaining similar detection performance. This architecture uses a backbone model as a base, as well as SSD. Also, to stabilize prediction scores, PPN uses a shared box predictor across different feature maps of different scales. Thus, this predictor takes into account all the training data, instead of independently using a portion of the training data for each predictor, as SSD does. Also, PPN uses max pooling operations to shrink the feature map down to 1×1. This operation does not have any addition and multiplication operations, and so is very fast. On the other hand, SSD uses the convolution operation to extract layers from the base network, and build the feature maps.

3.4. MobileNets

Google's USB Accelerator is fully compatible with 8-bit quantized MobileNets. Thus, in this work, several versions and variations of this DL-based model category are used. This set of models provide light-weight deep NN using depthwise separable convolutions. In other words, the model factorizes convolutions into depthwise, and 1×1 convolutions called pointwise convolutions. The first applies a single filter to the input channel, and the second applies a 1×1 convolution, combining the outputs of the first. The input of the CNN is a tensor with shape $D_f \times D_f \times M$, where D_f represents the input channel spatial width and height, and M is the input depth. After the convolution, a

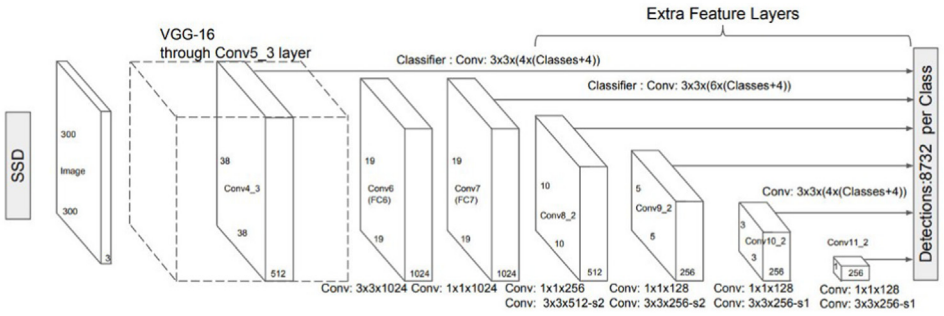

Fig. 2. SSD architecture (Liu et al., 0000).

feature map of shape $D_f \times D_f \times N$ is obtained, where N is the output depth. In this context, these families of models use two hyper-parameters that allow the user to resize the model so that it meets the system requirements. These hyper-parameters are: *width multiplier* α, and *resolution multiplier* ρ. The first is used to reduce the size of the CNN uniformly at each layer. For a given value of $\alpha \in (0, 1]$, the number of inputs channels M becomes αM, as well as the number of output channels N becomes αN. *Width multiplier* reduces the computational cost and the number of parameters by α^2. The second hyper-parameter, ρ, is also used to reduce the computational cost. This one is applied directly to the input image, setting its resolution. The values of $\rho \in (0, 1]$ are chosen in order to obtain typical input image resolutions. Similarly to the *width multiplier*, the *resolution multiplier* also reduces the computational cost and the number of parameters by ρ^2. So, both parameters are different ways of reducing the model size and computational cost. When combined, the effects on the final model can be even more significant.

3.5. Inception

Szegedy et al. (2015) proposed the primary version of Inception. This model design is based on the premise that the desired object to classify or detect can present several sizes on different images. This leads to the difficulty of choosing the right kernel size. To overcome this issue, Inception proposes three different convolutional filter sizes – 1×1, 3×3, and 5×5. Additionally, the NN model also computes max pooling. The output of all these operations is then concatenated, constituting the result of the respective Inception module.

Inception-V2 was developed in order to reduce the computational complexity of the original version. This is done by factorizing the convolution operations. For example, a 5×5 convolution is factorized into two 3×3 convolutions, improving runtime performance. In the same way, a $m \times m$ convolution can be factorized into a combination of $1 \times m$ and $m \times 1$ convolutions.

3.6. You only look once

YOLO (Redmon et al., 2016) is based on the Fully Convolution Neural Network (FCNN) architecture idea, that processes the input image once, returning its output prediction. This model divides the input image into a grid with $S \times S$ cells. Each cell that contains the center of an object is responsible for detecting it, producing a maximum of two bounding boxes and considering only one class. Each bounding box present in a given cell is associated with a score that reflects how confident the model is that the object lies inside the box, and how accurate it is predicting the object position. The model itself is composed of 24 convolutional layers followed by two fully connected layers. To improve the model, several versions of the original one were created. In particular, Tiny YOLO-V3 is a lite version of an enhanced version of YOLO, that allows increasing the inference frame rate, with a lower computational cost. In other words, this model provides a proper balance between accuracy and inference speed, overcoming the slower inference performance of YOLO-V3.

4. Experimental devices

In this work, two types of devices are used to perform real-time trunk detection. The first, Google's USB Accelerator, uses an Edge TPU to compute inference. The second, NVIDIA's Jetson Nano, uses its own GPU to do so.

4.1. Google USB accelerator

Google's Coral USB Accelerator (Fig. 3), provides an Edge TPU machine learning accelerator coprocessor. It is connected via USB to a host computer, allowing high-speed inference. This device is compatible with Tensorflow Lite, a lightweight version of TensorFlow designed for mobile and embedded devices, and can perform image classification, object detection, and semantic segmentation. To perform such tasks, the Edge TPU uses 8-bit quantized models. So, when training a 32-bit float model, it has to be quantized using either *quantization aware training* or *post training quantization* (Jacob et al., 2018). The first approach simulates the effect of 8-bit values during the training process using quantization nodes in the NN graph. The second does not modify the NN structure and is applied after training. However, it is less accurate than the first method. Alternatively, pre-trained models that are already quantized can be used if compatible with the Edge TPU. The device supports a range of operations and is most likely compatible with models designed for mobile devices, using the SSD architecture. After training the model, the *Edge TPU compiler* is used to assign inference operations to the device and the host CPU, as represented in Fig. 4. The compiled model differs from the original Tensorflow Lite model in the first operation of the graph. CPU processes the operations from the first non-supported operation of the TPU, until the end of the graph. The inference will be as faster as higher it is the number of operations assigned to the Edge TPU.

Fig. 3. Google Coral USB accelerator (Google et al., 2020).

4.2. NVIDIA Jetson nano

Jetson Nano Developer Kit (Fig. 5), provides a 128-core NVIDIA Maxwell @ 921 MHz GPU, capable of computing image classification, object detection, semantic segmentation, and speech processing. This device is compatible with the most popular machine learning frameworks, including TensorFlow. Additionally, Jetson Nano is compatible with an accelerator library called TensorRT. This Software Development Kit (SDK) allows high-performance DL inference and is compatible with TensorFlow and other common frameworks. The main focus of this tool is to optimize already trained NNs for a given purpose efficiently. To do so, TensorRT allows compressing the network after training. Also, this SDK optimizes the kernel selection and normalizes

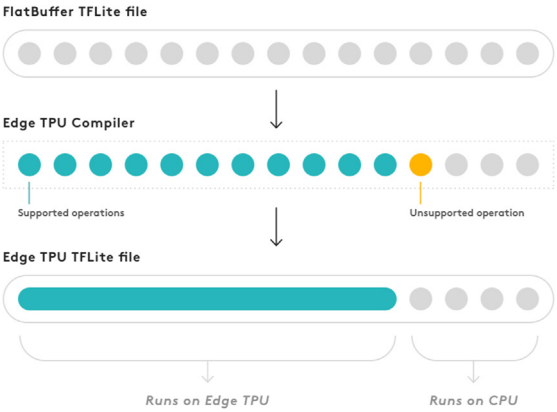

Fig. 4. Edge TPU model compilation scheme (Google et al., 2020).

Fig. 5. Jetson Nano development kit (Corporation et al., 2020).

and converts the model to a desired precision (32 or 16 floating-point (FP32/FP16), and 8 integer (INT8)) in order to improve latency, efficiency, and throughput.

5. Methods

In order to create a reliable vine trunk detector, in a first stage, a dataset was created using our robotic platform AgRob V16 (Santos et al., 0000), represented in Fig. 6. This dataset contains images captured in two different vineyards, each one with a camera with different characteristics. After data collection, the vine trunks were manually annotated using a graphical tool and converted to the Pascal VOC format (Everingham et al., 2010). The dataset containing the training images and the respective annotations is publicly available at our repository (http://vcriis01.inesctec.pt/ - `DS_AG_39`).

5.1. Data collection

As referenced before, two onboard cameras belonging to our robotic platform collected data in two different vineyards. Representative images of both vines can be observed at Fig. 7. One of the cameras is from Raspberry Pi and possesses a 640 × 480 resolution, and a conventional infrared filter (Fig. 7 at the right). The other is a Mako G-125C camera, with a resolution of 1292 × 964 resolution, and an infrablue filter (Fig. 7 at the left). The dataset is constituted by 336 different images and approximately 1600 annotated trunks. It contains:

- Images with two different resolutions. Two types of vine trunks, one with foliage, and other without.
- Trunks covered by shadows.

These conditions confer variety to the training procedure, and consequently, robustness to the inference final result.

Fig. 6. AgRob V16 robotic platform.

5.2. Data annotation

Given the training dataset, the perceptible vine trunks were manually annotated on the images. Fig. 7 shows an example of an image of each vine with the respective annotations. The output from this procedure is a set of bounding boxes with different sizes, for each image. These are represented in a `.xml` file with the Pascal VOC annotation syntax, containing the label class considered, and the four corners location of each bounding box. The annotations are also publicly available (http://vcriis01.inesctec.pt/) together with the training images. This data is the input for the training procedure, described below.

5.3. Training procedure

In the training procedure, two different frameworks were used. One was Tensorflow, where all the MobileNets and the Inception models were trained. On the other side, YOLO has its own training framework, also denominated YOLO. All the models used were pre-trained using the COCO dataset (Lin et al., 2014).

Fig. 8 demonstrates the step-by-step Tensorflow training procedure. This training flow converts raw bounding boxes annotations into the final models suitable for both the USB Accelerator and Jetson Nano. As referenced before, TL was used in order to retrain all the NN models used. To do so, firstly, the bounding boxes data was serialized into the *TFRecord* data type. This process stores data as a sequence of binary streams, allowing Tensorflow to interpret data in a more efficient manner. After retraining the models, in order to save them for posterior TL or retraining, they are exported into *frozen graphs*. These graphs can be exported for both devices used. For Google's USB Accelerator, the graph is converted to Tensorflow Lite and then compiled, as described in Section 3. If the pre-trained model is not 8-bit quantized, *post training quantization* is used to quantize it. For NVIDIA's Jetson Nano, the *frozen graph* was optimized and converted to TensorRT using 16-bit floating point precision. Table 1 shows all the models considered, with their respective variations. It is worth noting that two equal versions of MobileNet-V1 were retrained using different α values, in order to analyse the impact of this hyper-parameter.

For Tiny YOLO-V3, the pre-trained weights were also used as well as YOLO's own training framework. This training procedure is quite more straightforward, requiring only the pre-trained model, raw images, and bounding boxes as input, and no post-processing over the trained model.

6. Results

To evaluate the considered models in both USB Accelerator and Jetson Nano, a test subset containing 45 images and approximately 180 vine trunks was extracted from the training dataset. To evaluate and compare all the detectors in both devices, a state-of-the-art metric was used. Also, the runtime performance of the models on top of each device was measured.

6.1. Evaluation metric

The PASCAL Visual Object Classes (VOC) Challenge (Everingham et al., 2010) was used to evaluate the performance of the considered models, both on Google's USB Accelerator, and NVIDIA's Jetson Nano. This method is widely used to evaluated DL-based models performing object detection, being a fair approach to compare the performance of different models resulting from different works. Pascal VOC calculates the Average Precision (AP) as follows. Given an annotated *ground truth* bounding box B_g, and a detected bounding box B_d, the Intersection over Union (IoU) is computed using the following equation

Fig. 7. Images of the training data correspondent to the two different vineyards, and respective annotations.

$$IoU = \frac{m(B_g \cap B_d)}{m(B_g \cup B_d)} \tag{1}$$

where $m(x)$ denotes the area of x. Fig. 9, shows a graphical representation of this concept. So, IoU represents the quotient between the area of overlap and the area of union between the *ground truth*, and the detection bounding boxes. Using this definition, for a given threshold value t, three main concepts can be defined:

- **True Positive** (TP): IoU $\geqslant t$, i.e., a correct detection.
- **False Positive** (FP): IoU $\leqslant t$, i.e., an incorrect detection.
- **False Negative** (FN): a *ground truth* is not detected.

It is worth noting that if more than one detection for a single *ground truth* is computed, only the one with the highest IoU is considered as TP, and all the others are FPs. This being said, with these three qualifiers it is possible to define two fundamental concepts. The first, *precision* p, is defined as the total number of **TPs** over all the detections. The second, *recall* r, is the total number of **TPs** over all the *ground truths*. Using these, it is possible to plot a curve of the *recall* in function of the *precision*, $p(r)$. The evaluation considers that a suitable detector is the one that maintains the precision high for an increase in recall. With this consideration, the detector is evaluated computing the Average Precision (AP), interpolating the obtained curve, and calculating the area below the curve. Mathematically, this is expressed as follows

$$\sum_{r=0}^{1} \left(r_{n+1} - r_n \right) p_{interp}(r_{n+1}) \tag{2}$$

with

$$p_{interp}(r_{n+1}) = \max_{\tilde{r}:\tilde{r} \geqslant r_{n+1}} p(\tilde{r}) \tag{3}$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$.

For the sake of completeness, this work also evaluates the models using the F1 score. This score is the harmonic mean between the precision p and recall r, and can be calculated as follows

$$F1 = 2\,\frac{p \cdot r}{p + r} \tag{4}$$

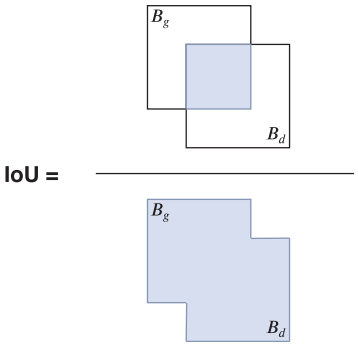

Fig. 9. Interception over union representation.

6.2. Detection performance

After training all the considered models, they were applied to the test dataset. Fig. 10 demonstrates an example of SSD MobileNet-V2 on both vineyards. For the Localization and Mapping purposes, the main interest is to find the presence of vine trunks in the image so that, then, using for example depth sensors, the 3D location of each can be extracted. So, in this work, we are not only interested in highly precise detections, but also in medium precise ones. Thus, using the metric previously described, a *IoU* threshold t equals to 0.5 was considered. In this way, the recall × precision curves were computed, and are represented on Fig. 11. Interpolating the curves, and calculating the area bellow them, the AP was computed for each model in each device. Additionally, using Eq. 4 the F1 score can also be computed for each model in each device. Table 2 summarizes this information, presenting

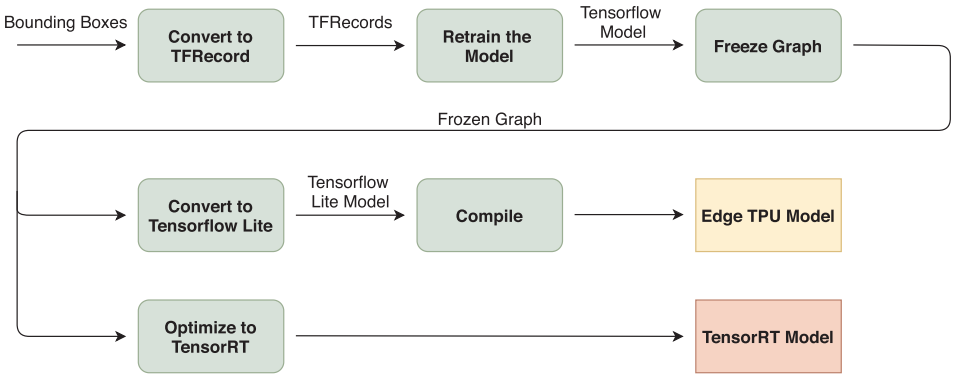

Fig. 8. Tensorflow-based training procedure flow.

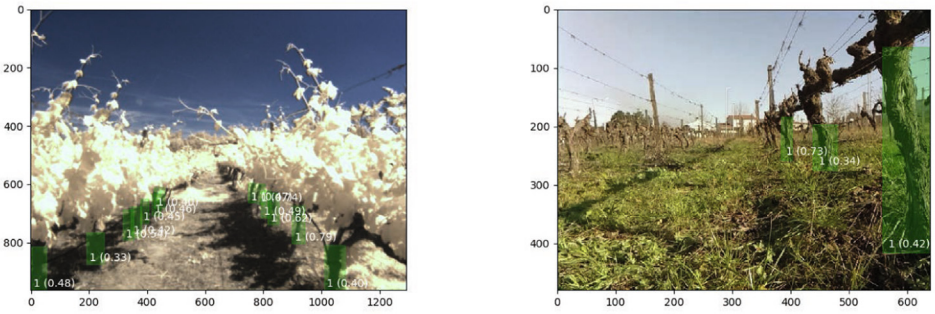

Fig. 10. An example of SSD MobileNet-V2 inference on both vineyards.

also the runtime performance of each configuration. To evaluate the runtime performance of each model in each one of the devices, all the inference configurations were profiled using the *high resolution clock* from `chrono` present in the *std* library. Table 2 contains the results of the profiling.

6.3. Discussion

By analysis of Fig. 11 and Table 2, it is visible that Tiny YOLO-V3 is only supported on NVIDIA's Jetson Nano. In fact, besides Tiny YOLO-V3, only SSD-based models are used (PPN and SSDLite are derivations of SSD) in this work. This is due to the incompatibility of different architectures such as Faster R-CNN (Ren et al., 2015), R-FCN (Dai et al., 2016), Mask R-CNN (He et al., 2017), and others, with Google USB Accelerator for object detection. This is a disadvantage in comparison with Jetson Nano, that supports a wider variety of architectures, and, consequently, models. Additionally, to perform inference on NVIDIA's Jetson Nano, *quantization* is not required. Inference is supported using both *floating-point* and *integer* precision on this device. On the contrary, the USB Accelerator only supports 8-bit quantized models. Despite this, Table 2 shows that Google's USB Accelerator has higher AP than Jetson Nano in all the SSD-based models. Starting from the same trained model, both devices optimize and convert it to better perform on each computational environment. The USB Accelerator converts the model to Tensorflow Lite and then compiles it, while Jetson Nano uses TensorRT to do so. Results show that the optimizations performed in the context of the USB Accelerator lead to a higher inference AP anf F1 score. Table 2 shows that, for example, SSD MobileNet-V1 presents an overall AP of 49.74% and F1 score of 0.61 on the USB Accelerator and AP of 41.39% and F1 score of 57% on Jetson Nano. Similarly, SSD MobileNet-V2 has higher AP on the first device than on the second, being 52.98% on the USB Accelerator, the higher AP obtained considering all the models in both devices. For SSD Inception-V2, the difference is quite significant, leading to the conclusion that TensorRT failed to optimize the model for Jetson Nano. On the Google USB Accelerator, this model achieves an F1 score of 61%, quite higher than the 19% score obtained on the Jetson Nano.

In terms of inference time, Table 2 also shows that generally, Google's USB Accelerator performs trunk detection at a higher frame rate than NVIDIA's Jetson Nano. For the MobileNets, the Google device has an average inference time per image in the range of 20.21–23.83 ms, while the NVIDIA device has a runtime performance in the range of 47.40–65.26 ms. This means that, for the MobileNets, the USB Accelerator is more than two times faster than Jetson Nano. Only for SSD Inception-V2, Jetson Nano is faster. This is due to two main reasons: the Inception-V2 model is much larger than the MobileNets, and the USB Accelerator is compatible with a lower number of operations of this model. Thus, when compiling it for the Edge TPU, these non-supported operations are assigned to the host CPU, which leads to an average inference time per image of 359.64 ms. Another important runtime consideration is the loading model time. Our experience showed that the USB Accelerator loads all the models much faster than Jetson Nano.

In a final note about the MobileNets, results show that the decrease of the hyper-parameter *width multiplier* leads to lower inference AP and F1 score and higher frame rate, as expected. Also, the PPN architecture confirmed to be a lighter version of the SSD, allowing the use of a higher resolution with no costs on the runtime performance. However, even using a much higher input image resolution, the model that used this architecture has shown an AP much lower than models that use the SSD architecture with lower resolutions. SSDLite MobileNet-V2, a lighter version of SSD MobileNet-V2 resulted, in this specific case, in a much least precise model, not showing any advantage in terms of runtime performance.

To summarize the comparison of both devices, Table 3 shows the pros and cons of each device on the topics addressed in this work. The "+" denotes for a positive behavior, and "−" for a negative/worse behavior or an unsupported feature.

6.4. Detector application

This work presents the first DL-based approach to detect vine

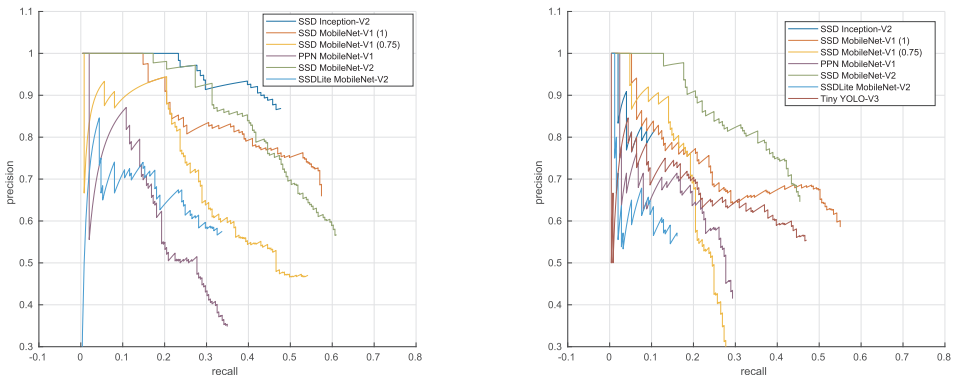

Fig. 11. Recall × precision curves of the models considered on (left) Google's USB Accelerator and (right) NVIDIA's Jetson Nano.

Table 2
AP (%), F1 score and average inference time per imgage (ms) obtained from the models considered using Google's USB Accelerator and NVIDIA's Jetson Nano.

Model	α	Resolution	Google USB Accelerator			NVIDIA Jetson Nano		
			AP (%)	F1 (%)	Runtime (ms)	AP (%)	F1 (%)	Runtime (ms)
SSD MobileNet-V1	1	300 × 300	49.74	61.00	21.18	41.39	57.00	56.83
SSD MobileNet-V1	0.75	300 × 300	39.78	50.00	20.21	21.95	23.00	47.40
PPN MobileNet-V1	1	640 × 640	23.17	34.00	21.83	20.38	34.00	53.23
SSD MobileNet-V2	1	300 × 300	52.98	59.00	23.14	40.08	53.00	62.84
SSDLite MobileNet-V2	1	300 × 300	23.64	42.00	23.83	10.78	25.00	65.26
SSD Inception-V2	1	300 × 300	46.10	61.00	359.64	9.45	19.00	73.71
Tiny YOLO-V3	–	416 × 416	–	–	–	32.56	51.00	100.2

Table 3
Summary of the pros and cons of each device.

	Google USB Accelerator	NVIDIA Jetson Nano
Compatibilitty	−	+
Floating-point support	−	+
Overall AP	+	−
Runtime performance	+	−
Time spent loading the models	+	−

Fig. 12. Vine trunk depth estimation using the proposed detectors.

trunks, that can be used in Localization and Mapping. Comparing with many state-of-the-art detectors for other applications, the ones proposed in this work presents lower AP and F1 score. However, in contrast with many works that use powerful GPUs, *low cost* and *low power* devices are used in this work. The models are 8-bit quantized on Google's Edge TPU and present 16-bit precision on Jetson Nano. This limits the inference performance in terms of detection AP. Even so, the detectors can be used in the Localization and Mapping procedures. Firstly, they perform inference at high frame rate. The navigation stack present on Agrob V16 (Fig. 6) imposes a minimum frame rate of ten *frames per second*. This condition is ensured, as demonstrated before. The stack uses a stereo camera system and builds a disparity map. In the context of the proposed work, recent experiments using Agrob V16 and deploying MobileNet-V1 on the vineyard in real-time shown that the proposed detectors can be used to extract vine trunks depth information using the disparity map. Fig. 12 shows an example of these experiments. The detections on the stereo images are projected into the disparity map. Calculating the median of all depths inside each bounding box results in the depth of each trunk, if the trunk occupies the majority of the region inside of the box. This result can be used to construct a map of the vineyard and, consequently, to localize the robot inside this map.

7. Conclusion

This work addresses the problem of feature extraction on steep slope vineyards for robotics localization and mapping. In this context, a reliable solution is required to detect vine trunks location on images. This work proposes the use of two different devices to do so: Google's USB Accelerator and NVIDIA's Jetson Nano. Seven different models were trained and deployed in these two devices. The performance of each one was analyzed both in terms of AP and runtime performance. Results showed that NVIDIA's Jetson Nano supports a wider variety of architectures and models than the USB Accelerator, that is limited to SSD-based architectures. On the other hand, for SSD-based models, Google's USB Accelerator provides higher AP and better runtime performance. In the best case, for SSD MobileNet-V2 on the USB Accelerator, an average precision of 52.98% and approximately 49 *frames per second* were obtained. Recent experiments demonstrate that, even so using these *low cost* devices, the proposed detectors can be used in Localization and Mapping.

In future work, the training dataset will be extended to use images from other vineyards, also considering thermal images. Also, a state-of-the-art DL-based model will be modified and adapted to perform better detecting vine trunks. Finally, DL-based semantic segmentation will be considered in order to segment the region occupied by the trunks.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded by the ERDF European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation — COMPETE 2020 under the PORTUGAL 2020 Partnership Agreement, and through the Portuguese National Innovation Agency (ANI) as a part of project "ROMOVI: POCI-01-0247-FEDER-017945". The opinions included in this paper shall be the sole responsibility of their authors. The European Commission and the Authorities of the Programme aren't responsible for the use of information contained therein.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, athttps://doi.org/10.1016/j.compag.2020.105535.

References

Andresen, T., de Aguiar, F.B., Curado, M.J., 2004. The alto douro wine region greenway, Landscape Urban Plann. 68 (2), 289–303, international Greenway Planning. doi: 10.1016/S0169-2046(03)00156-7. http://www.sciencedirect.com/science/article/pii/S0169204603001567.

Dos Santos, F.N., Sobreira, H., Campos, D., Morais, R., Moreira, A.P., Contente, O., 2016. Towards a reliable robot for steep slope vineyards monitoring. J. Intell. Robot. Syst. 83 (3–4), 429–444.

Roldán, J.J., del Cerro, J., Garzún-Ramos, D., Garcia-Aunon, P., Garzún, M., de León, J., Barrientos, A., 2018. Robots in Agriculture: State of Art and Practical Experiences. doi:10.5772/intechopen.69874. https://app.dimensions.ai/details/publication/pub.1100266943 and https://www.intechopen.com/citation-pdf-url/56199.

Duckett, T., Pearson, S., Blackmore, S., Grieve, B., Chen, W.-H, Cielniak, G., Cleaversmith, J., Dai, J., Davis, S., Fox, C., From, P., Georgilas, I., Gill, R., Gould, I., Hanheide, M., Hunter, A., Iida, F., Mihalyova, L., Nefti-Meziani, S., Neumann, G., Paoletti, P., Pridmore, T., Ross, D., Smith, M., Stoelen, M., Swainson, M., Wane, S., Wilson, P., Wright, I., Yang, G.-Z., 2018. Agricultural robotics: the future of robotic agriculture. arXiv:1806.06762.

Santos, F.B.N. d., Sobreira, H.M.P., Campos, D.F.B., Santos, R.M.P.M. d., Moreira, A.P.G. M., Contente, O.M.S., 2015. Towards a reliable monitoring robot for mountain vineyards. In: 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, pp. 37–43. doi:10.1109/ICARSC.2015.21.

Cheein, F.A., Steiner, G., Paina, G.P., Carelli, R., 2011. Optimized eif-slam algorithm for precision agriculture mapping based on stems detection. Comput. Electron. Agric. 78 (2), 195–207. https://doi.org/10.1016/j.compag.2011.07.007. http://www.sciencedirect.com/science/article/pii/S0168169911001542.

Aguiar, A., Santos, F., Santos, L., Sousa, A., 2019. Monocular visual odometry using fisheye lens cameras. In: Moura Oliveira, P., Novais, P., Reis, L.P. (Eds.), Progress in Artificial Intelligence. Springer International Publishing, Cham, pp. 319–330.

Jin, X.-B., Yang, N.-X., Wang, X.-Y., Bai, Y.-T., Su, T.-L., Kong, J.-L. Deep hybrid model based on emd with classification by frequency characteristics for long-term air quality prediction, Mathematics 8 (2). doi:10.3390/math8020214. https://www.mdpi.com/2227-7390/8/2/214.

Bai, Y., Jin, X., Wang, X., Su, T., Kong, J., Lu, Y., 2019. Compound autoregressive network for prediction of multivariate time series. Complexity.

Zhao, Z., Zheng, P., Xu, S., Wu, X., 2019. Object detection with deep learning: a review. IEEE Trans. Neural Networks Learn. Syst. 30 (11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444. https://doi.org/10.1038/nature14539.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning, MIT press.

Schmidhuber, J., 2015. Deep learning in neural networks: an overview. Neural Networks 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003. http://www.sciencedirect.com/science/article/pii/S0893608014002135.

Pan, S.J., Yang, Q., 2010. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191.

Weiss, K., Khoshgoftaar, T.M., Wang, D., 2016. A survey of transfer learning. J. Big Data 3 (1), 9. https://doi.org/10.1186/s40537-016-0043-6.

Lu, J., Behbood, V., Hao, P., Zuo, H., Xue, S., Zhang, G., 2015. Transfer learning using computational intelligence: a survey. Knowl.-Based Syst. 80, 14–23.

Shao, L., Zhu, F., Li, X., 2015. Transfer learning for visual categorization: a survey. IEEE Trans. Neural Networks Learn. Syst. 26 (5), 1019–1034. https://doi.org/10.1109/TNNLS.2014.2330900.

Torrey, L., Shavlik, J. Transfer learning, Handbook of Research on Machine Learning Applications doi:10.4018/978-1-60566-766-9.ch011.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C. SSD: single shot multibox detector, CoRR abs/1512.02325. arXiv:1512.02325. http://arxiv.org/abs/1512.02325.

Jin, P., Rathod, V., Zhu, X., 2018. Pooling pyramid network for object detection. arXiv:1807.03284.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C., 2018. Mobilenetv 2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510–4520.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2015. Rethinking the inception architecture for computer vision. arXiv:1512.00567.

Redmon, J., Farhadi, A., 2018. Yolov3: an incremental improvement. arXiv:1804.02767.

Joseph Chet Redmon, 2020. Yolo: Real-time object detection, https://coral.ai/products/accelerator, (accessed Jan 4, 2020).

Google, 2020. Usb accelerator, https://coral.ai/products/accelerator, (accessed Jan 15, 2020).

NVIDIA Corporation, 2020. Jetson nano developer kit, https://developer.nvidia.com/embedded/jetson-nano-developer-kit, (accessed Jan 17, 2020).

Santos, L., Santos, F.N., Oliveira, P.M., Shinde, P., 2020. Deep learning applications in agriculture: a short review. In: Silva, M.F., Luís Lima, J., Reis, L.P., Sanfeliu, A., Tardioli, D. (Eds.), Robot 2019: Fourth Iberian Robotics Conference. Springer International Publishing, Cham, pp. 139–151.

Kamilaris, A., Prenafeta-Boldú, F.X., 2018. Deep learning in agriculture: a survey. Comput. Electron. Agric. 147, 70–90. https://doi.org/10.1016/j.compag.2018.02.016. http://www.sciencedirect.com/science/article/pii/S0168169917308803.

Kamilaris, A., Prenafeta-Boldú, F.X., 2018. A review of the use of convolutional neural networks in agriculture. J. Agric. Sci. 156 (3), 312–322. https://doi.org/10.1017/S0021859618000436.

Dias, P.A., Tabb, A., Medeiros, H., 2018. Apple flower detection using deep convolutional networks. Comput. Ind. 99, 17–28.

Zheng, Y.-Y., Kong, J.-L., Jin, X.-B., Wang, X.-Y., Su, T.-L., Zuo, M. Cropdeep: The crop vision dataset for deep-learning-based classification and detection in precision agriculture, Sensors 19 (5). doi:10.3390/s19051058. https://www.mdpi.com/1424-8220/19/5/1058.

Koirala, A., Walsh, K.B., Wang, Z.-X., McCarthy, C., 2019. Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'mangoyolo'. Precis. Agric. 1–29.

Tian, Y., Yang, G., Wang, Z., Wang, H., Li, E., Liang, Z., 2019. Apple detection during different growth stages in orchards using the improved yolo-v3 model. Comput. Electron. Agric. 157, 417–426. https://doi.org/10.1016/j.compag.2019.01.012. http://www.sciencedirect.com/science/article/pii/S016816991831528X.

Bargoti, S., Underwood, J., 2017. Deep fruit detection in orchards, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3626–3633. doi:10.1109/ICRA.2017.7989417.

Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T., McCool, C. Deepfruits: A fruit detection system using deep neural networks, Sensors 16 (8). doi:10.3390/s16081222. https://www.mdpi.com/1424-8220/16/8/1222.

Kirk, R., Cielniak, G., Mangan, M. L*a*b*fruits: a rapid and robust outdoor fruit detection system combining bio-inspired features with one-stage deep learning networks, Sensors 20 (1). doi:10.3390/s20010275. https://www.mdpi.com/1424-8220/20/1/275.

Ding, W., Taylor, G.W., 2016. Automatic moth detection from trap images for pest management. Comput. Electron. Agric. 123, 17–28.

Zhong, Y., Gao, J., Lei, Q., Zhou, Y., 2018. A vision-based counting and recognition system for flying insects in intelligent agriculture, in: Sensors.

Steen, K.A., Christiansen, P., Karstoft, H., Jørgensen, R.N. Using deep learning to challenge safety standard for highly autonomous machines in agriculture, J. Imag. 2 (1). doi:10.3390/jimaging2010006. https://www.mdpi.com/2313-433X/2/1/6.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Lin, T.-Y., Maire, M., Belongie, S.J., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L., 2014. Microsoft coco: common objects in context. ECCV.

Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: towards real-time object detection with region proposal networks. arXiv:1506.01497.

Simonyan, K., Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556.

Deng, J., Dong, W., Socher, R., Li, L., Kai Li, Li Fei-Fei, 2009. Imagenet: a large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., 2015. Going deeper with convolutions. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Google, 2019. Tensorflow models on the edge tpu, https://coral.ai/docs/edgetpu/models-intro/, (accessed Jan 12, 2020).

Santos, L., Santos, F., Mendes, J., Costa, P., Lima, J., Reis, R., Shinde, P. Path planning aware of robot's center of mass for steep slope vineyards, Robotica 1–15 doi:10.1017/S0263574719000961.

Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A., 2010. The pascal visual object classes (voc) challenge. Int. J. Comput. Vision 88 (2), 303–338.

Dai, J., Li, Y., He, K., Sun, J., 2016. R-fcn: Object detection via region-based fully convolutional networks. arXiv:1605.06409.

He, K., Gkioxari, G., Dollár, P., Girshick, R., 2017. Mask r-cnn. arXiv:1703.06870.