FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Segmentation and detection of Woody Trunks using Deep Learning for Agricultural Robotics

Nuno Gonçalo Pinto Machado Namora Monteiro

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Armando Jorge Sousa

Second Supervisor: Dr. Filipe Neves dos Santos

July 30, 2020

Resumo

Os robôs agrícolas necessitam de algoritmos de processamento de imagem, que devem ser fiáveis em todas as condições meteorológicas e ser computacionalmente eficientes. Além disso, podem surgir várias limitações, tal como o *overfitting* no treino das redes neuronais que pode afectar o desempenho. Paralelamente, a evolução dos modelos de *Deep Learning* tornou-se cada vez mais complexa, exigindo uma complexidade computacional crescente. Assim, nem todos os processadores conseguem lidar de forma eficiente com tais modelos. Desta forma, o desenvolvimento de um sistema com um desempenho em tempo real para processadores de baixa potência torna-se exigente e é hoje um desafio de investigação e desenvolvimento pois há falta de *datasets* com anotações e ferramentas de agilização para apoiar este trabalho.

Para apoiar a implantação de tecnologia de *Deep Learning* em robôs agrícolas, no decurso deste trabalho foi desenvolvido o VineSet *dataset*, a primeira grande colecção pública de imagens de troncos de videiras. O *dataset* foi construído de raiz, tendo um total de 9481 imagens e respetivas anotações dos troncos da videira em cada um deles. O VineSet é composto por imagens RGB e térmicas de 5 vinhas diferentes do Douro, sendo 952 inicialmente recolhidas pelo robô AgRob V16, e as outras 8529 imagens resultantes de um vasto número de operações de augmentação.

Para verificar a validade e utilidade deste VineSet *dataset*, é apresentado neste trabalho um estudo experimental, utilizando modelos de *Deep Learning* do estado da arte com o *Google Tensor Processing Unit*. Para simplificar a tarefa de augmentação na criação de futuros *datasets*, propomos um procedimento de anotação assistida - utilizando os nossos modelos treinados - reduzindo o tempo de anotação num factor de dez vezes por *frame*. Esta dissertação apresenta resultados preliminares para apoiar futuras pesquisas neste tópico, por exemplo, o VineSet permite formar (usando *transfer learning*) redes neuronais profundas existentes com *Average Precision* (AP) superior a 80% para detecção de troncos de vinhas. Por exemplo, foi alcançada uma AP de 84,16% para o SSD MobileNet-V1. Além disso, os modelos treinados com o VineSet apresentam bons resultados noutros ambientes, tais como pomares ou florestas. A nossa ferramenta de anotação automática prova-o, reduzindo o tempo de anotação em mais de 30% em várias áreas da agricultura e em mais de 70% nas vinhas.

Nesta dissertação, propomos uma solução computacionalmente efeciente para a segmentação dos troncos das videiras. Primeiro, foram utilizados modelos de detecção de objectos em conjunto com o VineSet para efectuar a segmentação dos troncos. Para avaliar o desempenho dos diferentes modelos, foi construído um *script* que implementa algumas métricas de segmentação semântica. Os resultados mostraram que os modelos de detecção de objectos treinados com o VineSet não só eram adequados para a detecção do tronco como também para a segmentação do tronco. Por exemplo, foi atingido um *DICE Similarity Index* (DSI) de 70,78% para o SSD MobileNet-V1. Finalmente, a segmentação semântica também foi brevemente abordada. Um subconjunto das imagens do VineSet foi utilizado para o treino de vários modelos. Os resultados mostram que a segmentação semântica pode substituir modelos de detecção de objectos baseados em *Deep Learning* por modelos de classificação baseada em píxeis, se for fornecido um dataset para o

treino adequado.

Desta forma, todo o trabalho realizado permitirá a integração de algoritmos de *edge-AI* em sistemas de *Simultaneous Localization and Mapping* (SLAM), como o Vine-SLAM, que servirá para a localização e mapeamento do robô, através de marcadores naturais nas vinhas.

Abstract

Agricultural robots need image processing algorithms, which should be reliable under all weather conditions and be computationally efficient. Furthermore, several limitations may arise, such as overfitting in the training of neural networks that may affect the performance. In parallel with this, the evolution of Deep Learning models became more complex, demanding an increased computational complexity. Thus, not all processors can handle such models efficiently. So, developing a system with a real-time performance for low-power processors becomes demanding and is nowadays a research and development challenge because there is a lack of real data sets annotated and expedite tools to support this work.

To support the deployment of deep-learning technology in agricultural robots, this dissertation presents a public VineSet dataset, the first public large collection of vine trunk images. The dataset was built from scratch, having a total of 9481 real image frames and providing the vine trunks annotations in each one of them. VineSet is composed of RGB and thermal images of 5 different Douro vineyards, with 952 initially collected by AgRob V16 robot, and others 8529 image frames resulting from a vast number of augmentation operations.

To check the validity and usefulness of this VineSet dataset, in this work is presented an experimental baseline study, using state-of-the-art Deep Learning models together with Google Tensor Processing Unit. To simplify the task of augmentation in the creation of future datasets, we propose an assisted labelling procedure - by using our trained models - to reduce the labelling time, in some cases ten times faster per frame. This dissertation presents preliminary results to support future research in this topic, for example, with VineSet it is possible to train (by transfer learning procedure) existing deep neural networks with Average Precision (AP) higher than 80% for vineyards trunks detection. For example, an AP of 84.16% was achieved for SSD MobileNet-V1. Also, the models trained with VineSet present good results in other environments such as orchards or forests. Our automatic labelling tool proves this, reducing annotation time by more than 30% in various areas of agriculture and more than 70% on vineyards.

In this dissertation, we also propose the segmentation of the vine trunks. Firstly, object detection models were used together with VineSet to perform the trunk segmentation. To evaluate the performance of the different models, a script that implements some metrics of semantic segmentation was built. The results showed that the object detection models trained with VineSet were not only suitable for trunk detection but also trunk segmentation. For example, a DICE Similarity Index (DSI) of 70.78% was achieved for SSD MobileNet-V1. Finally, semantic segmentation was also briefly approached. A subset of the images of VineSet was used to train several models. Results show that semantic segmentation can substitute DL-based object detection models for pixel-based classification if a proper training set is provided.

In this way, all the work done will allow the integration of edge-AI algorithms in Simultaneous Localization and Mapping (SLAM) systems, like Vine-SLAM, which will serve for the localisation and mapping of the robot, through natural markers in the vineyards.

Acknowledgments

I would like to thank everyone that contributed to the concretization of this dissertation.

To Dr. Filipe Santos and Prof. Armando Sousa for the advice and support given all the way and for believing in me.

To André Aguiar especially, for all the help given, patience and friendship being always present in all moments of difficulty ensuring that problems were always overcome.

To my family for all the support, love, strength, and guide me always the best way during my academic walk.

To my colleagues and friends for their support over the last few years.

To my girlfriend for all the love and for calming me down at the hardest times.

Thank you.

Nuno Namora Monteiro

"The man who doesn't make up his mind to cultivate the habit of thinking misses the greatest pleasure in life."

Thomas Edison

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
AUC	Area Under the Curve
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DSI	Dice Similarity Index
DSSD	Deconvolutional Single Shot Detector
F1	F1-Scores
FLOP	Floating Point Operations Per Second
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
IoT	Internet of Things
JSI	Jaccard Similarity Index
mAP	mean Average Precision
ML	Machine Learning
NN	Neural Network
VOC	Visual Object Classification
R-CNN	Region-based Convolutional Neural Network
ROI	Region of Interest
RPN	Region Proposal Network
SSD	Simultaneous Localization and Mapping
SSD	Single Shot Multibox Detector
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit
YOLO	You Only Look Once

Chapter 1

Introduction

1.1 Context

In the past few years, robotics has evolved exponentially, introducing itself as a vital tool in the execution of repetitive tasks.

Robots appeared as a solution to many problems, among which stand out: on the one hand, the possibility to supplant the direct interaction of man in tasks where, in times, it was unreplaceable, contributing in this way to a marked decrease in work-related accidents; in other hand, robots gave way to the elimination of "dead times", due to their strong autonomy and ability to act for several hours straight [18]. Therefore, it can be said that, in addition to solving various problems, they ensure greater efficiency and accuracy in the tasks they perform. Thus, robotics has a definite impact on the development of autonomous, independent, robust and efficient systems.

Since the productivity of agriculture, for most of history, is mainly the result of human action, as it still is today, it is necessary to invest in new processes that face this problem.

In this sense, there has been an increase in the demand for robotic solutions to monitor and supervise agricultural crops [19]. Thus, new scientific fields arise, such as farming precision, also called digital agriculture, which boosts productivity and minimize environmental impact. As a basis, we have Machine Learning (ML), a mechanism that enables the machine to learn without being necessarily programmed, combined with new technologies and high-performance computing, which opens new horizons and makes work more efficient and effective [20].

In this context, one of the tremendous current challenges of robotics for agriculture is to achieve image processing algorithms that are robust to conditions of variable brightness and, at the same time, efficient and effective in order to be implemented in autonomous robots, of small dimension and with limited energy capacity.

Combining the needs and desires identified by the main associations of Portuguese farmers, the INESC TEC team from the Laboratory of the Center for Industrial Robotics and Intelligent Systems, has been developing a research project in the field of robotics for agriculture since 2014, called AgRob [21][22]. In the field of hillside agriculture, solutions have been developed for the four phases of agricultural robotization: monitoring, precision spraying, pruning and selective

harvesting. Over the years, the following robotic platforms have been created in this context: AgRob V14, AgRob V15 and AgRob V16.

This dissertation is part of the AgRob V16 platform project, which aims to evaluate the performance of machine intelligence algorithms for image processing and its automatic detection and segmentation of trunks in vineyards.

1.2 Motivation

This robotic platform is inserted in agricultural environments, in which specific factors, such as the changing light condition and the irregularities of high slopes of Douro Vineyards, are significant challenges to overcome. Currently, it has an image processing algorithm already implemented to detect elements that do not allow total accuracy and efficiency in all tests performed. In this way, this dissertation intends to study several Deep Learning (DL) based algorithms to be implemented, together with the most recent Google Tensor Processing Unit (Coral Edge TPU), in order to perform a comparative evaluation of the real-time performance of each of these. Also, this study will provide input for Simultaneous Localization and Mapping (SLAM) algorithms, which will serve for the location and mapping of the robot, through natural markers of vineyards.

The tests to be performed will be based on the dataset that will be created from the images existing in AgRob V16, to obtain a more variable dataset and a better accuracy. In short, it is intended that the study carried out will contribute to the development and improvement of current solutions in agriculture.

1.3 Objectives

The main objectives to be achieved in this dissertation are to analyse and evaluate the performance of the different DL-based algorithms, the respective benchmarking and the improvement of current solutions. For this, the main tasks defined to achieve these objectives are:

- Creation of the first dataset available to the scientific community related to the detection of trunks in the vineyards from the images captured by AgRob V16.
- Performance evaluation of the different DL-based algorithms for detecting vine trunks in the vineyard.
- Performance evaluation of the different DL-based algorithms for segmenting vine trunks in the vineyard.
- Implementation of the algorithms in Google's TPU.
- Creation of an assisted labelling procedure to reduce the time spent on data annotation.

1.4 Contributions

During this dissertation, the following contributions to the state of art were performed.

- VineSet dataset created accepted on the official ROS Agriculture community (http://wiki.ros.org/agriculture).
- Submitted paper "VineSet: A Deep Learning-Oriented Woody Crops Trunk Image Collection and an Assisted Labelling procedure" in Journal of Signal Processing Systems [in reviewing process].

This dissertation's main contributions constitute the creation of the first public large collection of vine trunk images, called VineSet. Moreover, this work provides a low-power and high-performance Deep Learning models for detection and segmentation of vine trunks, capable of executing real-time operations in agriculture robotics. Also, this work presents a creation of an assisted labelling procedure that aims to reduce the manual annotation time in several areas of agriculture.

1.5 Dissertation Rationale and Structure

In addition to this introductory chapter, the dissertation is composed of 6 other chapters. Chapter 2 intends to specify the problem description and the proposed solution. After that, Chapter 3 presents a brief introduction to the main concepts used in this work and chapter 4 presents the literature review where DL concepts and applications in agriculture are aborded. The description of the implementation comes in chapter 5. The work started by creating a dataset for vine trunks due to the non-existence of public datasets related to the detection of trunks in the vineyards. Then is aborded the implementation of various DL-based algorithms for detection and segmentation of vine trunks and their integration with Coral Edge TPU. After that, the development of the assisted labelling tool is addressed. This chapter is based on the article presented in the appendix A. Finally, the results and their discussion are presented in chapter 6 and the conclusions in chapter 7.

Chapter 2

Problem Statement and Direction

2.1 Problem Description

Some questions arise regarding the purpose of this dissertation:

- **Question 1:** How to deal with inherent problems such as overfitting in the training of neural networks and conditions of outdoor environments, e.g., lighting and terrain?
- **Question 2:** How to deal with the size and compatibility of the deep learning models to be implemented in the Google's processor?
- **Question 3:** Is it possible to combine real-time performance object detection with a low power processor?
- **Question 4:** Is it possible to use object detection models to preform segmentation?
- **Question 5:** How to deal with the high amount of time spent in manual annotations?

These five questions represent the main problems that this dissertation intends to answer. One of the significant challenges of agricultural robotics today is to achieve image processing algorithms that are robust to variable lighting conditions and at the same time efficient to be implemented in small autonomous robots with limited energy capacity. At this moment, the existing algorithm for trunk detection does not have the desired precision and do not use DL, sometimes having flaws due to the instability of the terrain of the vineyards or because inconstant lightness.

2.2 Proposed System Architecture

In order to answer and solve the issues mentioned above, the first step to be taken in this dissertation will be to obtain knowledge through the state of the art of the various DL architectures and the various DL models. Also, the respective benchmarks between them, present in the literature. Thereby, it is possible to choose the different models to be implemented that will later be analyzed as to their performance and to solve the different problems raised.

- **Question 1:** Overfitting is one of the frequent problems in ML that consists of modelling the data too well, learning only the expected output for each input instead of learning the general distribution of the input data. Usually, one of the ways to detect overfitting is to see if the validation loss rises during the train. In addition to this problem, we have conditions such as lighting or the terrain of outdoor environments that may affect performance. In order to solve and minimise this problem, two solutions are proposed. The first is the use of data augmentation techniques, e.g., rotation and translation, in order to increase and make the dataset as varied as possible. The second is to analyse and do an early stopping if the validation loss rises during the train.

- **Question 2:** With the evolution of DL algorithms, they became more and more complex, demanding an increased computational complexity. In order to find the appropriate solutions to implement, it is necessary to analyze the different benchmarking of the DL models present in the literature. Moreover, since Edge TPU is used, only TensorFlow Lite models that are entirely 8-bit quantized are compatible. Thus, it is necessary to opt for models with greater precision and model complexity supported by the processor.

- **Question 3:** Not all processors can handle the complexity of DL models. Thus, the combination of real-time performance object detection with a low power processor is not always achieved. In order to solve this problem, it is necessary to choose a processor capable of high-performance machine learning, in this case, the proposed processor will be the Coral Edge TPU.

- **Question 4:** Usually, the object detection models have as main objective the detection of the object itself. However, we propose the hypothesis of using these models to segment an object. For that, the proposed solution is to make the annotations of the smaller bounding boxes in order to fill the various pieces of the trunk structure. Additionally, a preliminary approach is made to the use of semantic segmentation models to perform the segmentation.

- **Question 5:** One of the most expensive tasks in the creation of the datasets is the time spent on manual annotations. In order to reduce this time, the solution proposed is to create an assistance tool capable of using the models trained in object detection and through the process of inference extract the respective annotations.

In summary, this dissertation consists of a system that combines a set of DL models and architectures for both detection and segmentation that will be implemented in Coral Edge TPU through the TensorFlow framework. Also, this work proposes the use of DL models to create an assisted labelling tool. Figure 2.1 represents the described system architecture.

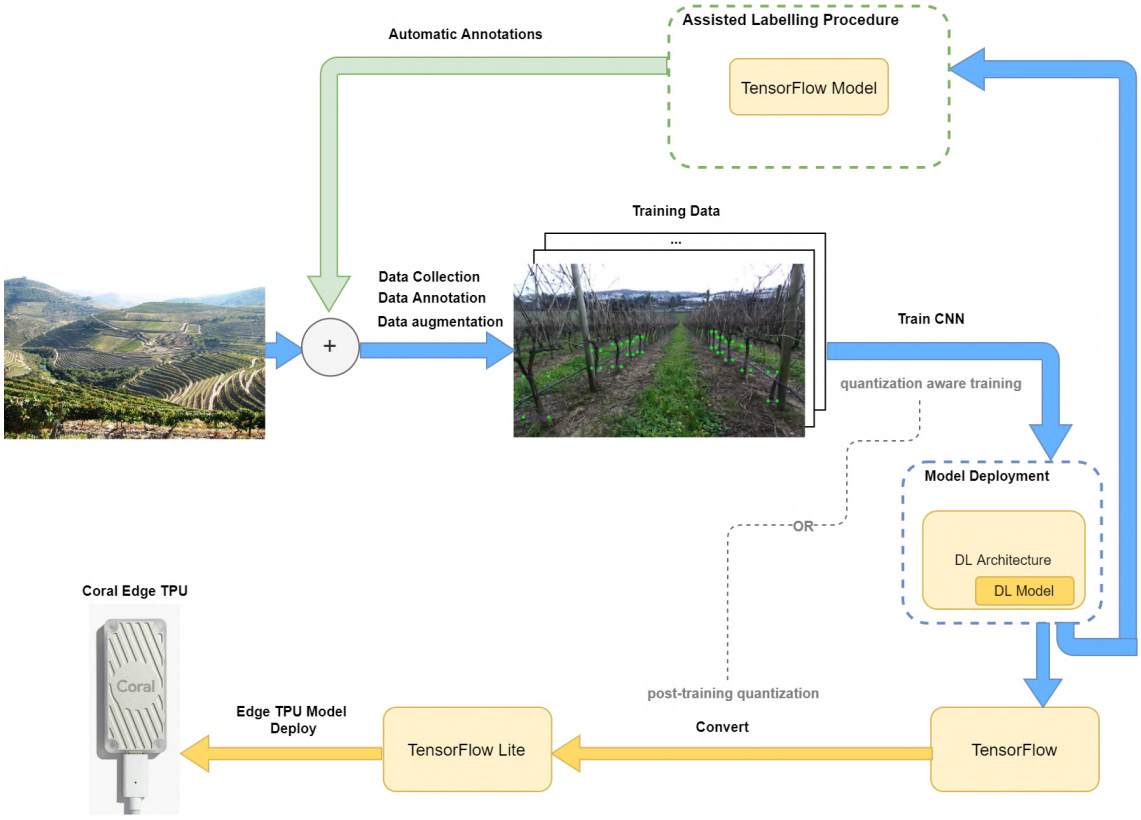


Figure 2.1: Global system architecture

Chapter 3

Fundamentals

This chapter aims to introduce the main concepts that will be used to describe the processes used in the implementation of this dissertation. Firstly, the main concepts for training Convolutional Neural Networks (CNN) are aborded. Finally the metrics for object detection and semantic segmentation are described.

3.1 Training

In DL the training phase is where the model parameters θ are optimised to minimise the cost function. However, several concepts arise throughout the training and can interfere with the performance and compatibility of the model. Thus several concepts are here briefly described.

3.1.1 Generalisation

Generalisation [1] refers to the capacity of an ML model to perform well on previously unseen data.

In the training phase of the DL model, the measured error, called the training error, is used to optimise the θ parameter to minimise the cost function. After the training phase is completed, a different dataset is used than the training dataset, i.e. different from the training set, called the test set. This will be used to measure how well the ML model can generalise to data not seen by the model when it was learning. The error associated with this measurement is called generalisation error, also called test error.

In this context, the main objectives for the ML algorithm to be successful is to minimise the training loss during the training phase and to minimise the gap between training and test loss. These two points correspond to two of the more significant challenges in machine learning, underfitting and overfitting.

In one hand, underfitting occurs when the model is not complex enough to capture the pattern in the training data, i.e., the model is not able to obtain a sufficiently low error value on the training set. On the other hand, overfitting occurs when a model is too complex and learns the training data

too well instead of learning the general distribution of the data, i.e., the difference between the training error and test error is too high.

Overfitting, underfitting and appropriate model complexity are visualized in Fig. 3.1.

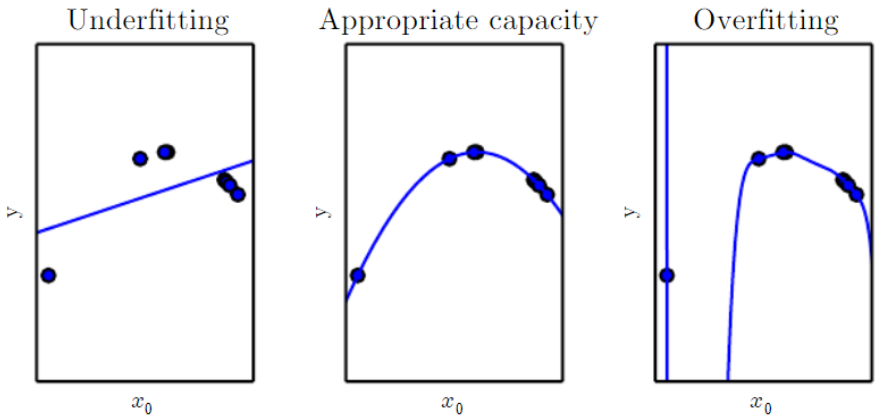


Figure 3.1: Set of three models that explicit the case of underfitting, generalisation and overfitting [1].

3.1.2 Transfer Learning

In the training process, it is unusual to train a CNN from scratch, not only due to the need to have a relatively large dataset to obtain good results, but also because days or weeks of training are required in Graphics Processing Unit (GPU) clusters.

Thus, a commonly used solution is the concept of Transfer Learning (TL) that aims to extract knowledge from one or more domains and applies that knowledge to another domain [2], as represented in Fig. 3.2.

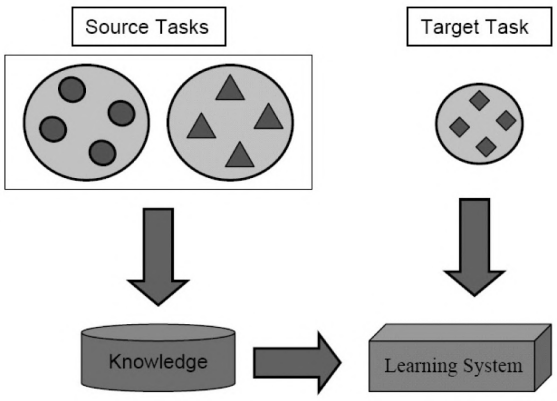


Figure 3.2: Learning process of transfer learning [2].

In this way, pre-trained networks are used as initialization or feature extractor for the task to be implemented, enabling to utilize knowledge from previously learned tasks and apply them to newer. Therefore, TL can be done in two ways, last layers-only retraining or full model retraining.

In one hand, as described in Fig. 3.3, the retraining of the last layers consists primarily of using a pre-trained network that has been trained in the source task, e.g., ImageNet [23], which consists of a large dataset of labelled images. In this way, this pre-trained network can be used as a feature extractor, removing the last fully-connected layer and then treat the rest of the base CNN as a fixed feature extractor for the new dataset. It is not only being transferred the parameters to the target task but also the addition of an adaptation layer, the fully-connected layer, in order to compensate for the different aspects of the images of the source and the target as described on the figure. During the training, only the weights of the last few layers of the model will be updated, where the final classification occurs. The rest of CNN remaining as a fixed extractor feature. This approach can be made with a smaller dataset [3].

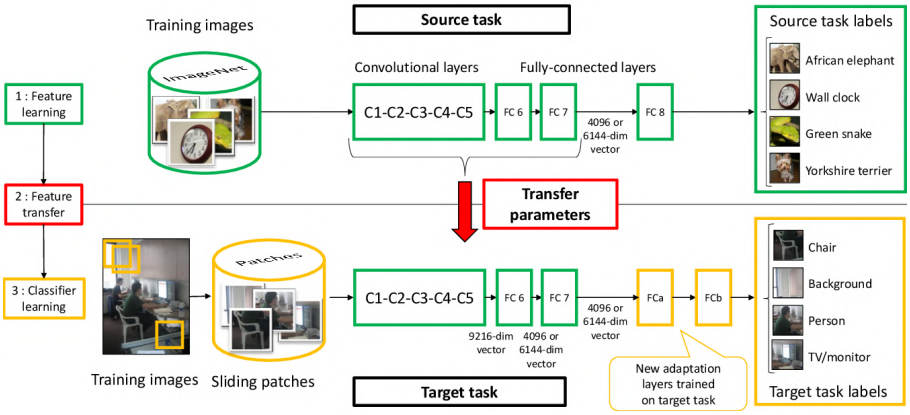


Figure 3.3: Tranferring parameters of a CNN [3].

On the other hand, retrain the full model consists of using the pre-trained network to initialise all the weights adjusting all the weights of each layer during the train. This method requires a dataset of a significant sample size to avoid overfitting [3].

3.1.3 Cost Function

The cost function can be viewed as the loss or error of the model and is used to evaluate the performance of the model. The goal of training is to learn the parameters θ, i.e., the values of the weights which can minimize the cost function and leads to improved model performance [1]. The function cost generally decomposes as a sum over training data of some loss function, as described in equation 3.1.

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}} L(x,y,\theta) \tag{3.1}$$

Where $J(\theta)$ is the cost function and L is the loss function. One loss function for example is the negative log-likelihood, also called cross-entropy loss. So in this case $L = -\log p(y|x;\theta)$, and result on the following equation,

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}} \log p_{model}(y|x) \tag{3.2}$$

3.1.4 Hyperparameters

Controlling algorithm's behavior can be done in the most of ML algorithms with hyperparameters. Unlike the parameters that are updated by the learning algorithm itself, e.g., the parameter θ, the values of hyperparameters are not optimized during the training phase but are configured before the learning process begins [1].

One example of hyperparameter is the learning rate (η) for example in a mini-batch stochastic, as described on equation 3.3, that specifies the speed of the weight (θ) updates in the learning phase.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3.3}$$

Where $J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$ is the cost function between the training set observations $x^{(i:i+n)}$ and the corresponding estimated response $y^{(i:i+n)}$.

Many more examples of hyperparameters exist. For example, the number of iterations of the training, the input resolution of the images and the number of classes, i.e., how many different types of objects exist. Other examples are the selection of the base network, the batch size, i.e., the number of samples that go through the network in each iteration, and the fine-tune for TL. Also, the optimizer can be changed, for example, momentum, which is a gradient descent optimization algorithm that aims to decrease oscillations represented as follows.

$$\begin{aligned} \upsilon_t &= \gamma \upsilon_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - \upsilon_t \end{aligned} \tag{3.4}$$

Where υ_t is the gradient vector, γ is the momemtum value, η is the learning rate and $J(\theta)$ is the cost function.

3.1.5 Quantization

Model quantization [17] consists of converting all the 32-bit floating-point numbers, like weights, to the closest 8-bit fixed-point numbers making the model smaller and more efficient. This process makes the TensorFlow models compatible with Edge TPU, which uses 8-bit quantized models, and can be done in two ways, quantization-aware training and full integer post-training quantization.

3.1.5.1 Quantization-aware training

This approach requires an initial modification to the network before the training simulating the impact of 8-bit numbers throughout training by using quantization nodes in the neural networks

(NN) graph. The fact that the 8-bit weights are learned during the training rather than being converted later like post-training quantization results in a higher accuracy model. Besides, it supports more operations [17].

3.1.5.2 Full integer post-training quantization

This second approach does not demand an initial modification to the NN structure and is applied after training to convert a pre-trained network into a quantized model. Nevertheless, this implies a representative dataset with the same data range that can be the dataset previously used for training, which is an essential point to identifying a precise 8-bit representation of each weight and activation value [17].

3.2 Metrics

In order to evaluate the various DL models performance to detect and segment the trunks in vineyards, the PASCAL Visual Object Classes (VOC) Challenge [24] metrics and F1-Score [25] were used and are here briefly described.

3.2.1 Intersection Over Union

Intersection Over Union (IOU), also known as the Jaccard Similarity Index (JSI), is an essential metric in deciding the object prediction of deep learning models. This measure evaluates the overlap between two bounding boxes [26], i.e., the area of ground truth and predicted area to the total area as represented in equation 3.5.

$$JSI = \frac{|A \cap B|}{|A \cup B|} \tag{3.5}$$

This metric is commonly used in two different computer vision tasks of object recognition that is object detection and semantic segmentation.

For object detection, IOU is given by the intersection area between the predicted bounding box B_p and the ground truth bounding box B_{gt} divided for the area of union between them [26].

$$IOU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{3.6}$$

The Fig. 3.4, illustrates the IOU between the ground truth bounding box, green, and detected bounding box, red. Therefore, we can verify if a detection is valid, i.e, true positive, or, not valid, i.e, false positive [26].

For semantic segmentation, the term JSI is more commonly used. However, the concept is the same as the IOU. For this method, pixels are used, instead of bounding boxes. So, JSI metric measures the intersection of the predicted segmentation pixels and the ground truth pixels, divided by the union [24].

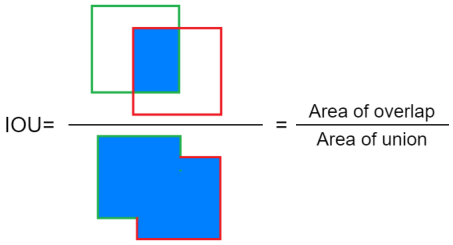


Figure 3.4: IOU for object detection.

Another way to calculate IOU or JSI [24] is using True and False Positives and Negatives definition in section 3.2.2, that results on follows equation:

$$JSI = \frac{TP}{TP + FP + FN} \tag{3.7}$$

3.2.2 True Positive, False Positive, False Negative and True Negative

Metrics for detection and segmentation problems involve correct and incorrect decisions. The terms true positive (TP), false positive (FP), true negative (TN) and false negative (FN) are frequently used in the evaluation of detection and segmentation results for evaluation the agreement or disagreement between the result of the prediction and the ground truth, as shown in table 3.1. Usually is set a threshold greater than or equal to 50% to IOU in order to verify if it is a correct detection or not [26].

Positive / Negative: refers to the decision made by the detection algorithm;

True / False: refers to how the decision agrees with the ground truth.

True Positive (TP): for object detection refers to a correct detection, for semantic segmentation refers to a correct association of a predicted pixel with the respective class, i.e, IOU≥Threshold;

False Positive (FP): for object detection refers to a wrong detection, for semantic segmentation refers to a wrong association of a predicted pixel with the respective class, i.e, IOU≤Threshold;

False Negative (FN): for object detection refers to a ground truth not detected, for semantic segmentation refers to a ground truth pixel with no associated prediction pixel;

True Negative (TN): for object detection refers to all bounding boxes that were correctly not detected, for semantic segmentation refers to a pixel that is correctly identified as not belonging to the given class.

		Predicted	
		Positive	**Negative**
Labeled	**Positive**	TP	FP
	Negative	FN	TN

Table 3.1: Classification of a detection with True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN).

3.2.3 Precision

From the definition of true/false positives/negatives, several metrics emerge. One of them is precision, that is the ability of a model to identify only the relevant objects [26]. It measures the fraction of predictions classified as positive that are really positive, and it is given by:

$$Precision = \frac{TP}{TP+FP} = \frac{TP}{all\ detections} \tag{3.8}$$

3.2.4 Recall

Recall is the ability of a model to find only all relevant cases correctly classified, i.e, all ground truth bounding boxes [26]. It measures the fraction of positive predictions that are correctly labeled and its given by:

$$Recall = \frac{TP}{TP+FN} = \frac{TP}{all\ ground\ truths} \tag{3.9}$$

3.2.5 Precision versus Recall curve

The Precision versus Recall curve [26] is a good way to evaluate the performance of an object detector since it allows to observe how precision and recall alter as confidence changes with the curve for each class of object. The x-axis of this curve represents the recall, and the y-axis represents the precision, where each point on the curve constitutes recall and precision for a specific confidence value.

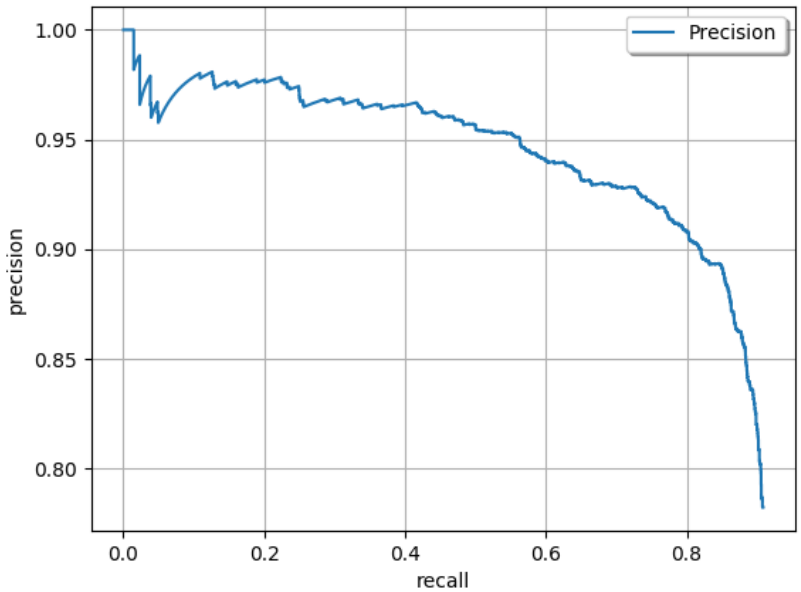


Figure 3.5: Precision versus Recall curve.

An ideal model would have high precision when recall increases, meaning that if the confidence threshold varies, the precision and recall continue to be high. Otherwise, the model performs

poorly. So, a weak object detector needs to enhance the number of detected objects, by reducing the confidence threshold, to detect all ground-truth objects correctly [26].

However, this curve is typically noisy with a saw-tooth shape resulting from the trade-off between precision and recall. Due to this fact, it is tough to evaluate the performance of the model and compare various models with their precision versus recall curves crossing each other.

3.2.6 F1-Score

The F1-Score (F_1), also known as the Sørensen–Dice coefficient or Dice Similarity Index (DSI), aims to measure the harmonic mean between precision and recall and can range between 0 and 1 [27][28]. This metric reaches the best value at 1, implying a perfect precision and recall. So, F1 summarizes model ability for a particular value of the threshold and is given by the following equation:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.10}$$

For semantic segmentation, the term DSI is more commonly used [29][30]. However, the concept is the same. So, DSI described in equation 3.11 consists on two times the number of pixels common between the prediction P_p and ground truth P_{gt} masks divided by the total number of pixels in both masks.

$$DSI = \frac{2|P_p \cap P_{gt}|}{|P_p \cup P_{gt}|} \tag{3.11}$$

The DSI equation 3.11 is very similar with the JSI equation 3.2.1, since they are positively correlated through the following equations:

$$JSI = \frac{DSI}{2 - DSI} \Leftrightarrow DSI = \frac{2\,JSI}{1 + JSI} \tag{3.12}$$

Another way to calculate F_1 or DSI is using True and False Positives and Negatives definition in section 3.2.2, that results on follows equation:

$$DSI = \frac{2TP}{2TP + FP + FN} \tag{3.13}$$

3.2.7 Average Precision

Precision versus Recall curves typically have a zigzag pattern and usually cross each other, which makes it difficult to compare the different curves on the same plot. In this way, instead of comparing curves, it is beneficial to have just a number that describes the performance. So, a standard metric is the Average Precision (AP) [26] that can be described at the following equation.

$$AP = \int_0^1 p(r)dr \tag{3.14}$$

In a simple way, the AP can be described as the precision averaged over all recall values between 0 and 1, i.e., the area under the curve (AUC).

Primordially, the Pascal VOC Challenge used 11-point interpolation. However, the AP value is currently calculated using all recall levels to improve the comparison with low AP score algorithms, as stated in [24].

11-Point Interpolation

The AP 11-Point Interpolation summarises the shape of the precision versus recall curve and is stipulated through an average of precision at a set of eleven equally spaced recall levels $[0, 0.1, ..., 1]$, through the following equation

$$AP = \frac{1}{11} \sum_{r \in [0, 0.1, ..., 1]} p_{interp}(r) = 1 \tag{3.15}$$

Therefore, the AP is calculated by interpolating the precision merely at the 11 levels r by taking the maximum precision measured where $\widetilde{r} > r$, as we can see in the equation 3.16, where p($\widetilde{r}$) is the measured precision at recall r.

$$p_{interp}(r) = \underset{\widetilde{r}, \widetilde{r} \geq r}{max}\, p(\widetilde{r}) \tag{3.16}$$

Interpolating all points

On the other hand, interpolation through all points can be described by the following equation:

$$AP = \sum_{r=0}^{1} (r_{n+1} - r_n) p_{interp}(r_{n+1}) \tag{3.17}$$

Now the AP value is accessed by interpolating the precision at each level r, by taking the maximum precision measured where $\widetilde{r} > r_{n+1}$ and p($\widetilde{r}$) is the measured precision at recall r, as we can see on the equation 3.18.

$$p_{interp}(r_{n+1}) = \underset{\widetilde{r}, \widetilde{r} \geq r_{n+1}}{max}\, p(\widetilde{r}) \tag{3.18}$$

Thus, the AUC can now be calculated through all recall points. This area results in the sum of all areas separated by a fall of precision at a particular recall r under the curve.

Chapter 4

State of Art

This chapter presents the state of art of the main concepts present in the dissertation. Firstly, it describes the main concepts of ML and its classes. Secondly, the various existing deep learning architectures are addressed. Then, the various architectures and models of deep learning are analyzed and compared in order to solve the main problems in agriculture in terms of accuracy and complexity to detect objects. Subsequently an approach is made to the platforms that will be used, both in terms of hardware and framework. Finally, a brief description of the main DL applications on agriculture is presented.

4.1 Machine Learning

Machine learning has arisen with big data technologies and high-performance computing to create new horizons in the agriculture domain [4]. This leads to new concepts like precision farming and smart farming [31], that improves production efficiency.

An ML model aims to solve problems in a repeatable way making autonomous decisions about the information that is present in datasets by identifying patterns and getting knowledge from the data available and predicting future instances under a task performed in previous observations.

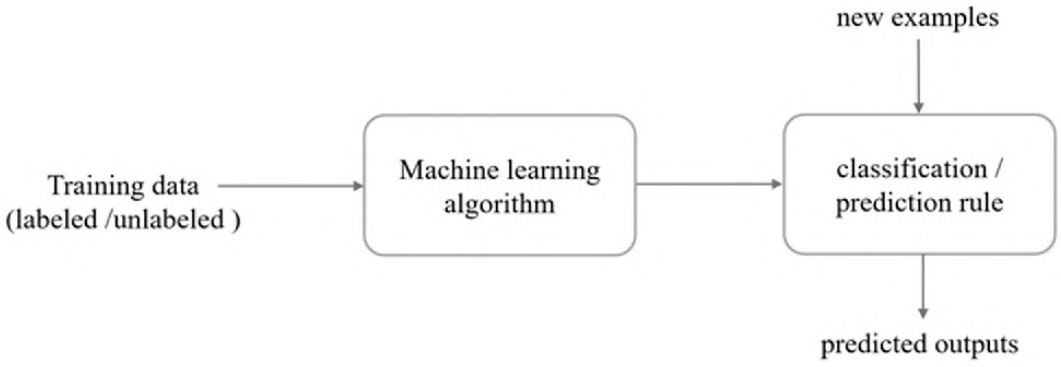


Figure 4.1: ML approach [4]

The methodology implemented by ML as represented in Fig. 4.1, implies a learning process intending to learn from experience, i.e., training data, to perform a task with as minimum human intervention as possible. The output is made after the learning process ends, that consists of predictions of an ML that was not programmed explicitly. This ML model runs an algorithm that adaptively improves its performance by learning from experience over time, like humans.

4.1.1 Deep Learning

Deep learning is a sub-field of machine learning, extending classical ML by adding more complexity into the model [25], which attempts to learn high-level abstractions in data through hierarchical architectures [7]. The popularity of DL techniques has increased and implemented in several agricultural areas in the last years [32]. Kamilaris et al. [33][25] introduce some applications of DL applications on agriculture, even though not considering hardware requirements imposed by the complexity of the DL models. Convolutional Neural Networks (CNN) constitutes a class of DL, and they are the preferred technique when it comes to feature extraction for images in DL.

DL and CNNs have been remarkably widespread in the field of computer vision. Therefore, CNNs are commonly used for several computer vision tasks such as Object Detection and Semantic Segmentation [8]. In the following sections, these topics and various DL models and architectures will be briefly described, since they constitute a fundamental part of this dissertation.

4.1.2 Neural Networks

Neural Network (NN) is a class of models within the general ML literature. NNs are a biologically-inspired programming paradigm which allows a computer to learn from observational data, while DL is a robust set of techniques for learning in NN [34].

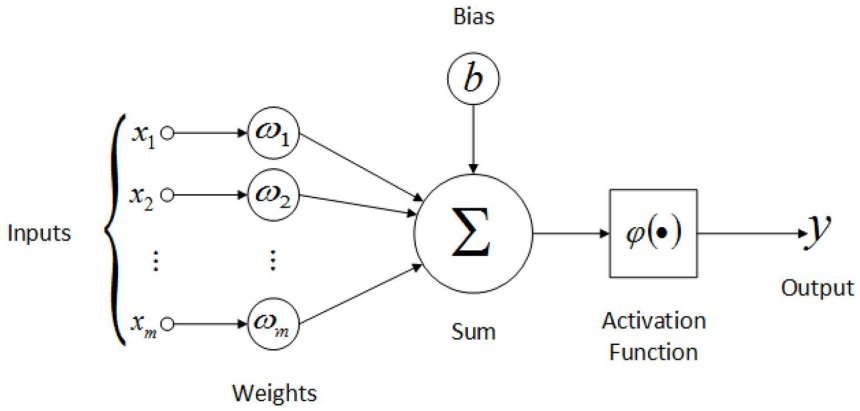


Figure 4.2: McCulloch-Pitts model of a neuron[5]

By observing Fig. 4.2, the neuron receives x_j as input signal where $j \in \{1,2,...,m\}$. Each input has a corresponding weight of w_j. Afterwards, the inputs and weights are linearly combined to compute a weighted sum of this m inputs, followed by an activation function φ, that defines the output of the neuron, as we can see on the following equation 4.1.

$$y = \varphi \sum (x_j w_j + b) \tag{4.1}$$

4.2 Object Detection

Object detection aims to define the bounding boxes around each object of a specified class in an image. The detection bounding box is set as a true positive if it overlaps more than the defined threshold with the ground truth box, that is normally 50% [35]. As Fig. 4.3 shows, a bounding box typically symbolize the location of an object.

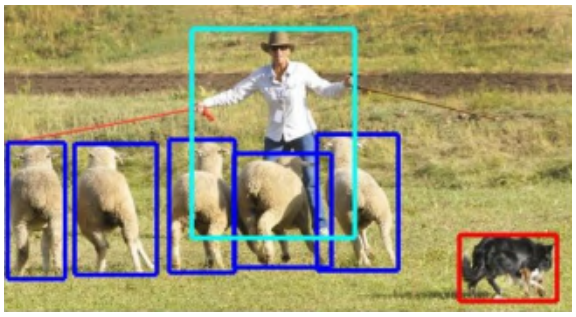


Figure 4.3: Generic object detection bounding boxes example [6].

4.2.1 Convolutional neural networks

CNNs constitutes a class of DL, forming a variant of feed-forward neural networks [36], that is used to resolve computer vision challenges [32]. Over time they have achieved great prominence in the field of image classification since Krizhevsky et al. [37] won the ImageNet Large Scale Visual Recognition Challenge in 2012. Thenceforward, they appear in numerous of the surveyed papers as the technique used [25].

As explained in [9], CNN has multiple advantages over the traditional methods, some of them are:

- The highly hierarchical structure;
- Deeper architectures provide a significant increase capability when compared with conventional models;
- CNN architecture allows to improve related tasks together, combining classification and bounding box regression into a multi-task on Fast R-CNN;
- From a different viewpoint can solve several traditional computer vision challenges recast as high-dimensional data transform problems.

In addition to the advantages listed above, CNN allows parallelization as a result of learning complex problems relatively rapid due to weight sharing [32].

As Fig. 4.4 shows, the architecture is composed by the convolutional layer, the pooling layer and the output layer that is normally a fully connected layer. Several convolutions are performed, generating distinct representations of the dataset, starting more generic at the first larger layers and ending more specific at the deeper layers [25].

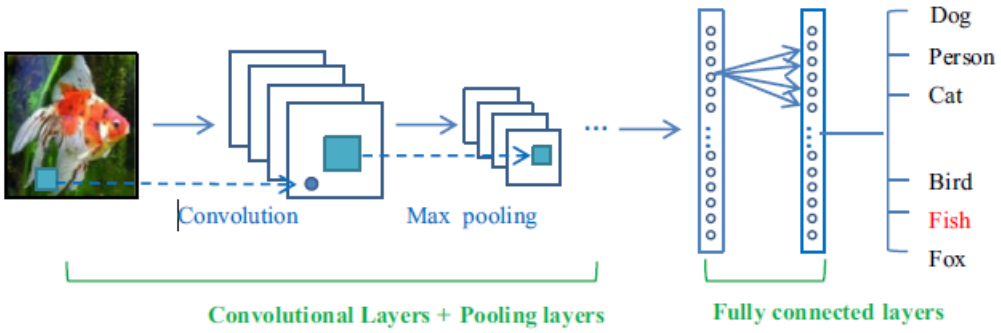


Figure 4.4: Example of an CNN Architecture Model [7]

The convolutional layers apply the convolution operation, as the follow equation 4.2:

$$s(t) = (x * w)(t) \tag{4.2}$$

where $*$ represents the convolution operator and x can be represented as the input of the convolutional layer and w the output, designated by kernel or feature map. Adapting equation 4.2 to convolution operation, if we use a two-dimensional image I as input at position (i, j) and also a two-dimensional output kernel K, results on equation 4.3:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{4.3}$$

This operation is applied to the input, acting as feature extractors from the input images whose dimensionality is reduced by the pooling layers [33]. The fully connected layers, placed in many cases near the output of the model, is responsible for converting the two-dimensional feature maps obtained from the previous layers into a one-dimensional vector and act as classifiers exploiting the high-level features learned, with the purpose of classifying the input images into their respective classes or to making numerical predictions [25].

Generally, providing an adequate large dataset, CNN can enhance the probability of correct classifications. For that, data augmentation is a technique usually used to increase the dataset and improve CNN accuracy. Making it robust even under challenging conditions such as orientation, illumination, different resolutions or complex backgrounds, thus enabling increase the dataset by implementing random transformations to the original images [25][32].

4.3 Semantic Segmentation

In contrast, semantic segmentation classifies each pixel in an image and estimate the probability of that pixel belongs to a specific class [8]. However, the output does not distinguish between the same class. Usually, the standard metric used to evaluate this type of models is the intersection over union over all the predicted pixels and ground truth pixels of the entire dataset [35]. Fig. 4.5 represents an example of a semantic segmentation ground truth.

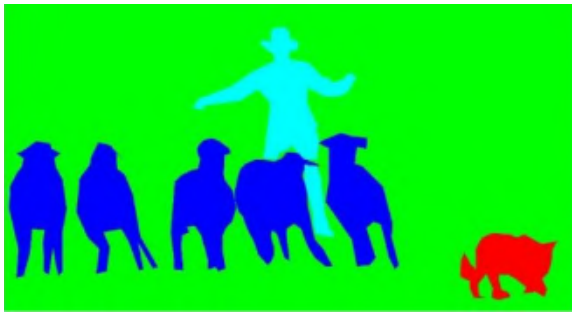


Figure 4.5: Semantic Segmentation mask example [6].

4.3.1 Convolutional neural networks for semantic segmentation

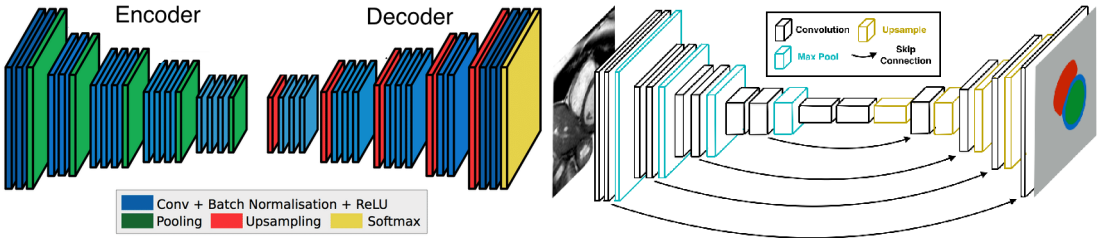
In CNNs for semantic segmentation [8], the initial layers are responsible for learning the low-level concepts like edges and colours, and the last level layers learn the high-level concepts like the shape and size, i.e., the different objects. So, initially, the neurons contain the information for a small region of the image. As we move through the layers into the high-level neurons, the image size keeps on decreasing, the channels keep on increasing, and the neurons contain information for a vast region of the image. In this case, the pooling layers perform the downsampling.

In object detection, fully connected layers are needed since it is required to map the spatial tensor from the convolution layers. However, it destroys the spatial information [8].

Conversely, in semantic segmentation, no fully connected layers are used, since we need to retain the spatial information. In this way, the convolutional layers associated with downsampling layers produce a low-resolution tensor containing the high-level information. So, taking the high-level information, we have to produce high-resolution segmentation outputs by adding more layers associated with upsampling layers. As the resolution increases, the number of channels decreases, thus accessing low-level information [8].

This whole process is part of an encoder-decoder structure, as we can see in Fig. 4.6a, where the layers that downsample the input belong to the encoder and the layers upsample to the decoder [8].

To summarize, the encoder outputs a tensor containing information about the objects, and then the decoder takes this information and produces the segmentation maps [8]. However, some low-level information can be lost. Due to this, the decoder needs to access the low-level features



(a) (b)

Figure 4.6: Encoder-Decoder Architectures with (a) no skip connections, and with (b) skip connections [8].

produced by the encoder layers, and that is achieved by adding skip connections like Fig. 4.6b describes.

4.3.2 Object Detection using convolutional neural networks

Generic object detection consists of two main topics, the region proposal based and the regression/classification based [9].

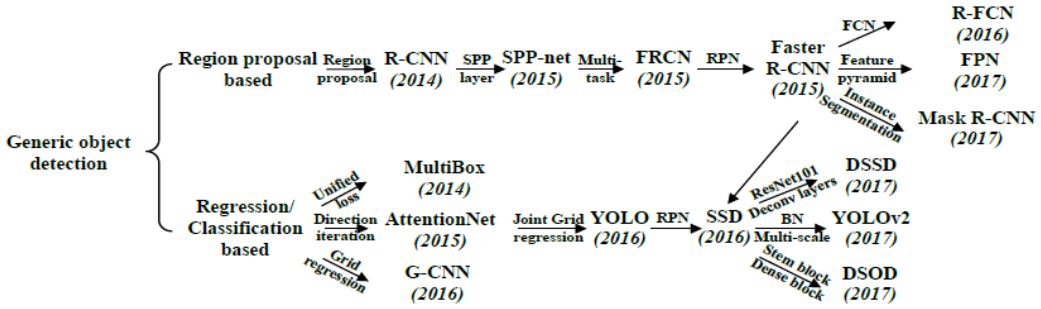


Figure 4.7: Generic object detection [9]

4.3.2.1 Region-based Convolutional Neural Network

The R-CNN was described in 2014 by Girshick et al. [10]. This model consists of 3 modules. The first denominated as Region Proposal, generate and extract category independent region proposals, these proposals establish the set of candidate detections, for example, candidate bounding boxes. The second denominated as Feature Extractor is a large CNN that extracts features from each candidate region. The third module denominated as Classifier, classify features using, for example, linear Support Vector Machine (SVM), as one of the known class. This module can use different classification methods. Since Regions of Interest (ROIs) are identified, a standard classification can be used to identify and label different objects.

A method called *Selective Search* use propose candidate regions or bounding boxes of potential objects in the image, combining the best intuitions of segmentation and exhaustive search. From

segmentation, the sampling process is used. In the other hand, exhaustive search is used to capture all possible object locations. However, other region proposal methods are allowed due to the flexibility of R-CNN [38].

The main goal is to take an image and correctly identify where the objects are through bounding boxes in the image. R-CNN architecture is shown in Fig. 4.8.

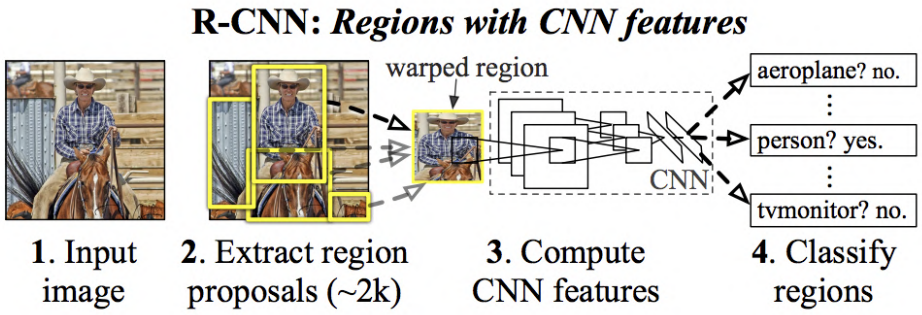


Figure 4.8: R-CNN Architecture [10]

4.3.2.2 Fast R-CNN

Due to the limitations of R-CNN Girshick [11] in 2015 proposed an improvement to solve speed problems resulting in the Fast Region-Based Convolutional Network (Fast R-CNN).

The main limitations of R-CNN can be summarized as follows:

- Training is a multi-step pipeline: preparation and operation of three separate models.
- Training takes up a lot of time and space: training with very deep networks on several region proposals is very slow.
- Low speed of object detection using deep networks to provide predictions on several region proposals is very slow.

For this, Fast R-CNN is used as a single model instead of a pipeline to learn and classify the originate regions. Fast R-CNN architecture is illustrated in Fig. 4.9.

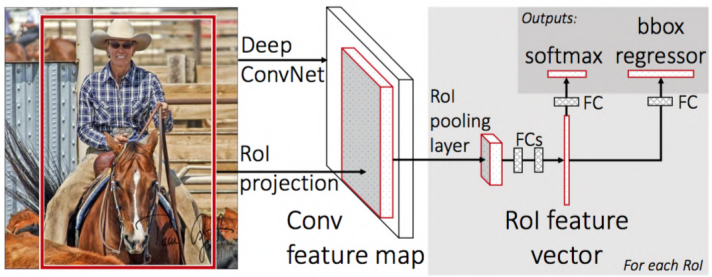


Figure 4.9: Fast R-CNN Architecture [11]

The architecture takes as input an entire image a set of region proposals. The first step is to process the whole image passed through a deep CNN. For feature, extraction is used a pre-trained CNN. Then exists a layer called ROI Pooling Layer, that extracts a fixed-length feature for a provided input candidate region. The fully connected layers are used to interpret the output of the CNN, followed by a division into two outputs per ROI, one to predict the class of the proposed region via softmax layer, the other to the bounding-box regression offsets [11].

4.3.2.3 Faster R-CNN

Ren et al. [39] proposed a new architecture that results in Faster R-CNN. Although it is a single unified model, is composed of two modules. The first module, Region Proposal Network (RPN), a CNN that proposes regions, and the second module is the Fast R-CNN that extracts features from the proposed regions and makes the bounding box and respective class labels.

The main idea is that both modules work on the same output of the CNN, making the RPN module tells the Fast R-CNN module where to look. To summarize, the Faster R-CNN is a combination between the RPN and the Fast R-CNN architecture, both sharing the convolutional layers between the two networks, minimizing the time spent identifying the ROIs with the selective search method, replacing it with the RPN [39].

4.3.2.4 You Only Look Once

Redmon et al. [12] in 2015 proposed the YOLO architecture, a new approach for object detection. Initially, divides the input image into a grid of cells, an SxS grid. If the centre of a bounding box falls into a grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes that involves the x, y coordinate, the width(w), the height(h), the respective confidence scores for those boxes and predicts C class probabilities [40], as described in Fig. 4.10,. All these predictions are encoded as a tensor that can be calculated as follow equation 4.4.

$$S \times S \times (B * 5 + C) \tag{4.4}$$

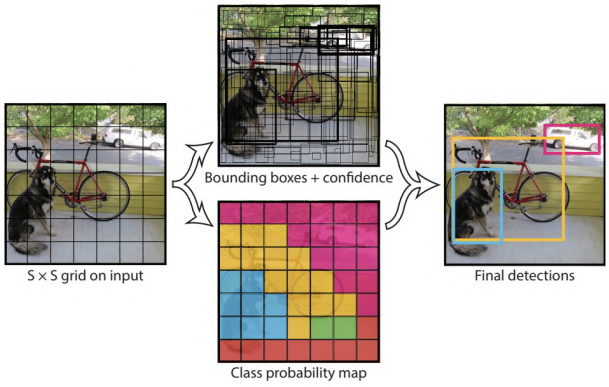


Figure 4.10: YOLO Model [12]

Nevertheless, this method has some limitations. So, Redmon et al. [41] proposed some improvements in order to overcome these constraints. YOLO method use fully connected layers on top of the extracted feature map to predict the coordinate of bounding boxes.

Instead of this, YOLOv2 [41] is inspired on the work done in Faster R-CNN in the RPN, removing the fully connected layers and the bounding box prediction was made with the use of anchor boxes. The same authors proposed an improvement in YOLOv2, thus increasing the accuracy, giving rise to YOLOv3 [42].

4.3.2.5 Single Shot Multibox Detector

The SSD method, likewise to the YOLO method mentioned before, consists of predicting all the bounding boxes at once and classify the result of those detections. Therefore, Redmon et al. [12] say that YOLO imposes some limitations in dealing with small objects in groups due to spatial constraints on bounding box predictions. Since each grid cell can only predict two boxes and can only have one class, as we can see on Fig. 4.10. By that, this method has conflicts to generalize to objects in new aspect ratios or configurations.

Liu et al. [13] aiming at these difficulties proposed a Single Shot MultiBox Detector (SSD), in which most of the techniques were adopted from the MultiBox method [43]. The core idea of this algorithm is to train a convolutional network that outputs the coordinates of the bounding boxes directly, the height and width of the box. Simultaneously, it produces a vector of probabilities corresponding to the confidence over each class of objects.

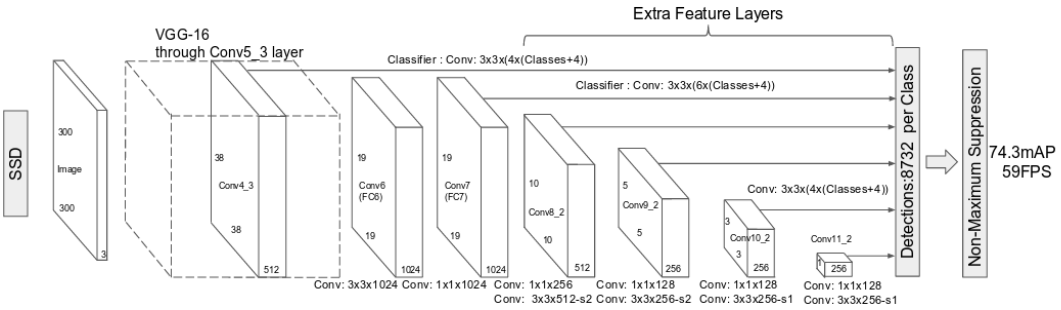


Figure 4.11: SSD Architecture [13]

The first architecture layers, as shown in Fig. 4.11 are based on a standard model used for high-quality image classification, in this case, the VGG16 network was used as the backbone on this SSD architecture, but other networks should also produce good outcomes. On the end of this model, several feature layers are added to predict the several bounding boxes with different feature maps resolutions, aspect ratios and respective confidences. The NN is trained with a weighted sum of localization loss, like Smooth L1, which is the loss between the predicted box and the ground truth, and confidence loss, like Softmax, which is loss over multiple classes confidences [9]. The training step is crucial to have better accuracy, for this we need to choose a set of default boxes

and scales for detection as well as hard negative mining and data augmentation strategies, thus improving the performance of the SSD model.

To summarize, the SSD model is based on a feed-forward convolutional network that generates a fixed-size of bounding boxes and rates them in conformity with the presence of object class instances in those boxes. The final detection results by a non-maximum suppression step, keeping the highest-rated bounding boxes [13].

4.3.3 Comparison between Deep Learning Architectures

The region proposal architectures R-CNN [10], Fast R-CNN [11] and Faster R-CNN [39] use a two-shot approach, one to create the regions where objects are expected to be and the other shot to detect objects in those regions. This approach usually has a better accuracy but a high inference time. Contrarily, single-shot architectures like YOLO [12] and SSD [13] can find all objects within an image in one shot, making them more efficient and has good accuracy.

Usually, all the works [10], [11]–[13] evaluate the efficiency of the different models through a standard performance metric, the mAP, which is the mean Average Precision. Besides, they also use PASCAL Visual Object Classification (PASCAL VOC) [24], this being one of several datasets for object detection, classification and segmentation. This dataset is commonly used in object detection competitions.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Figure 4.12: Results on Pascal VOC2007 test [13]

The results presented in Fig. 4.12 represents the performance reported in [13], demonstrate the precision of the different models and their speed. With these results, the SSD algorithm stands out as the better candidate for a real-time obstacle detector algorithm. However, other DL architectures are evolving, giving rise to new solutions such as YOLOv3 [42] our DSSD [44].

4.3.4 Comparison between Deep Learning Models

Computational cost impacts on the recognition accuracy studies are not abundant in the literature, Canziani et al. [45] in 2016 proposed an analysis of some DNN architectures by implementing tests on an NVIDIA Jetson TX1 board. It was measured several performance indices like accuracy rate and model complexity, but the key of these analyses was the relationship between these

performance indices. Although it is a valuable work, it has been dedicated to a few numbers of DNNs and only tested on NVIDIA Jetson TX1 board.

For that reason, Bianco et al. [14] in 2018 provided a more comprehensively study over 40 different DNN architectures for image recognition analyzing elements such as computational cost and accuracy on two distinct hardware platforms, NVIDIA Jetson TX1 and NVIDIA Titan X.

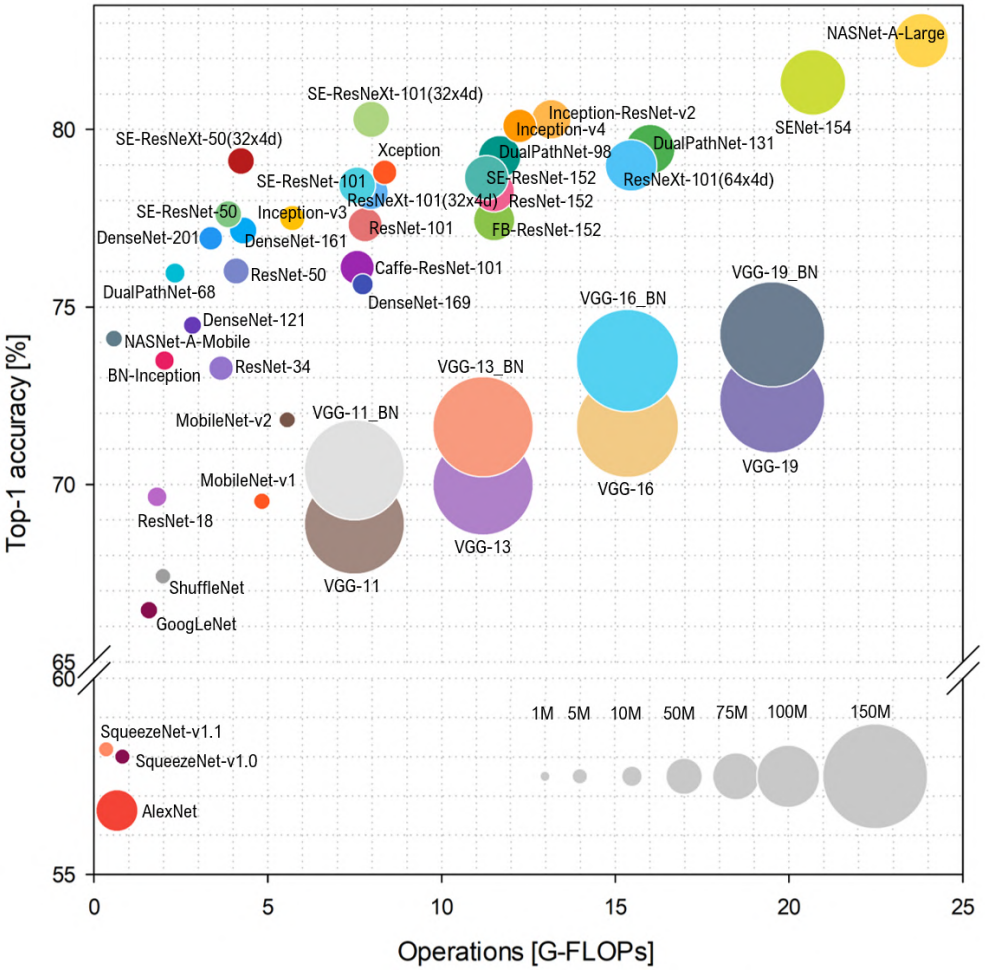


Figure 4.13: Top-1 accuracy vs floating-point operations (FLOPs) [14]

As represented in Fig. 4.13, the size of each ball corresponds to the model complexity

Top-1 is the traditional accuracy, that means the model answer should be precisely the expected answer, and Top-5 accuracy represent any model that gives five higher probability answers that shall correspond to the expected answer [32]. So, by analysis of Fig. 4.13, we can see that the DNN model reaching the highest accuracy is the NASNet-A-Large [46]. However, it also has the highest computational complexity. Between all the models that have the lowest level of model complexity, SE-ResNeXt-50 [47] has the highest accuracy and low level of model complexity.

From Fig. 4.14 we observe how efficient the parameters of each model are used and can notice that SqueezeNets [48], ShuffleNet [49], the MobileNets [15] and NASNet-A-Mobile [46], are the models that use their parameters more efficiently.

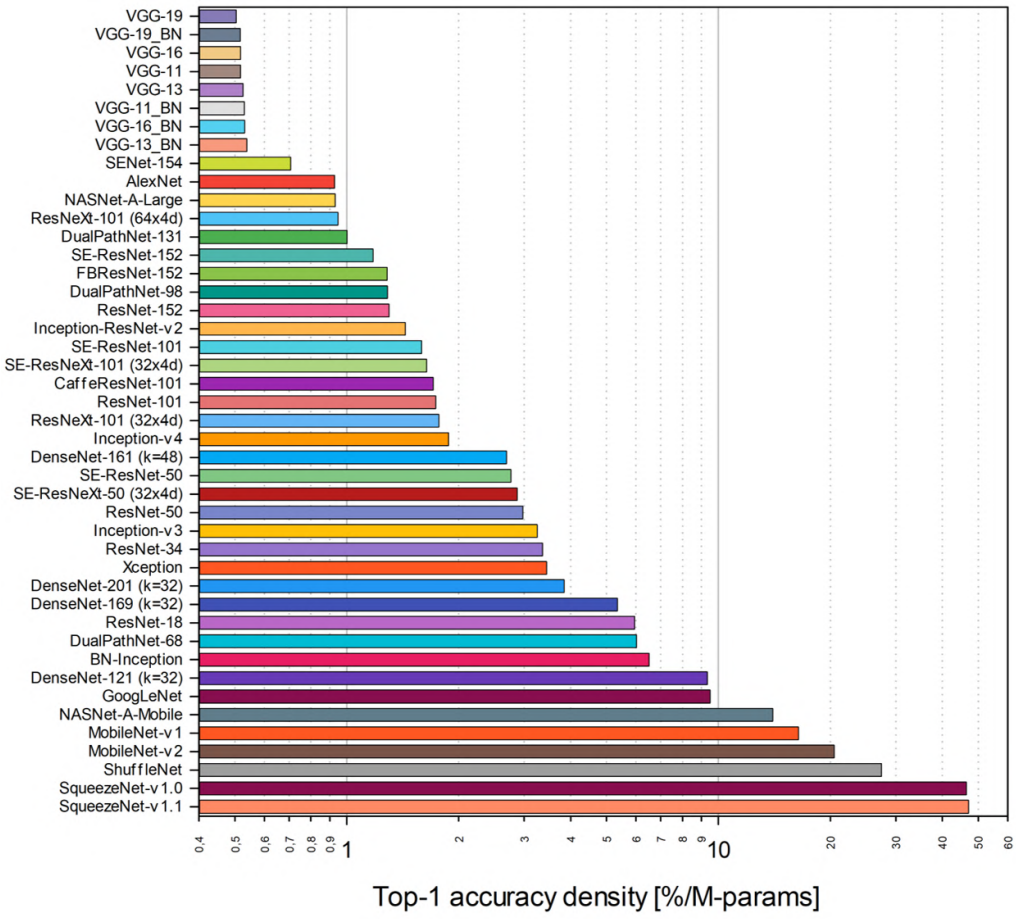


Figure 4.14: Top-1 accuracy vs. Top-1 accuracy density [14]

From this study, we can extract the information that if the number of operations increase, does not imply that the detection accuracy increases too. Also, not whole the DL models use parameters with the same level of efficiency. Thus, this comparison among different DL models provides us with knowledge for the right choice of architecture.

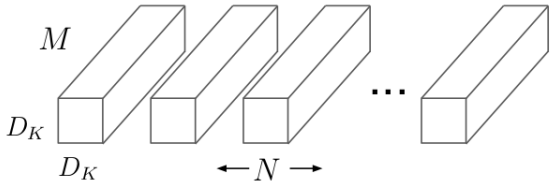
4.3.5 MobileNet

MobileNets [15] provides lightweight DL models oriented to use in embedded, low cost and power systems, using depthwise separable convolutions that consists of factorising the standard convolution into a depthwise convolution and a 1x1 convolution called pointwise convolution. So, the first apply a single filter to each input channel and then is followed by a pointwise convolution to combine the outputs of depthwise convolution.

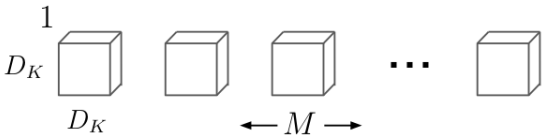
The input of a CNN is a $D_f \times D_f \times M$ feature map. After the convolution, produces a $D_f \times D_f \times N$ feature map, where D_f is the spatial width and height of a square feature map, M is the number of input channels (input depth), and N is the number of output channels (output depth).

The standard convolutional layer is parameterised by convolution kernel of size $D_k \times D_k \times M \times N$ where D_k is the size of the kernel, as we can see on Fig. 4.15, where M and N have the

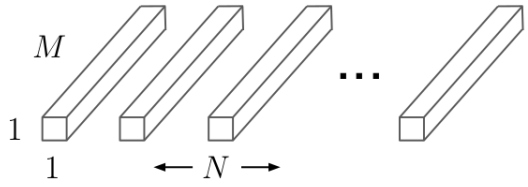
same meaning as described above.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 4.15: MobileNets convolution architecture. The standard convolution is divided into two layers, a depthwise and a pointwise convolution, that combined result on the depthwise separable convolution [15].

Therefore, standard convolutions have the computational cost of:

$$D_k \times D_k \times M \times N \times D_f \times D_f \tag{4.5}$$

The cost associated with depthwise separable convolution consists of the sum of the cost of the two layers. The depthwise convolution has a cost of:

$$D_k \times D_k \times M \times D_f \times D_f \tag{4.6}$$

The pointwise convolution has a cost of:

$$M \times N \times D_f \times D_f \tag{4.7}$$

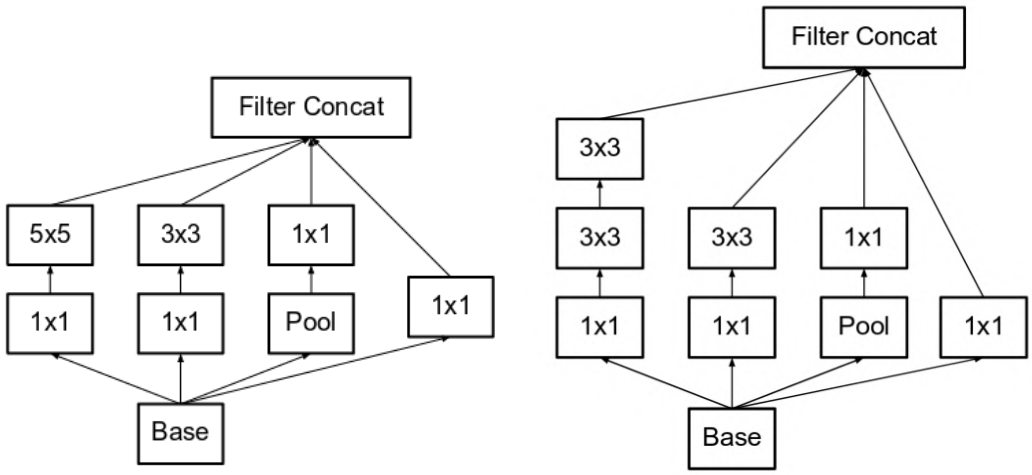
In this way, the depthwise separable convolutions cost is:

$$D_k \times D_k \times M \times D_f \times D_f + M \times N \times D_f \times D_f \tag{4.8}$$

Using 3x3 depthwise convolutions, MobileNet uses 8 to 9 times less computation than standard convolution, which is evident in inference time. In the task of object detection, the results for MobileNet trained on the COCO dataset [6] prove that it remains the smallest and least computationally expensive base network compared to VGG and Inception for both the SSD and Faster-RCNN framework while having similar or superior AP.

4.3.6 Inception

The Inception model [16] came with the same objective as MobileNet, reducing the model size and the respective computational cost in low-power processors. So, this model scaled up by factorising convolutions and adding regularisation.



(a) (b)

Figure 4.16: Different versions of Inception module, (a) the original Inception module and (b) Inception module where 5×5 convolution is replaced as two 3×3 convolutions [16].

The Inception as the name implies uses Inception modules which uses convolutions with multiple filters sizes, executing these in parallel, and the outputs are concatenated and sent to the next inception module. These modules are modified throughout the various versions of the model to increase the speed of the network, where more extensive convolutions are replaced by multiple smaller ones as seen in Fig. 4.16, which reduces the number of parameters due to the weight sharing between adjacent tiles [16]. Comparing with MobileNet, the Inception use standard convolution instead of depthwise convolutions. This results in a lesser number of parameters in MobileNet. However, this results in sometimes in a slight decrease in the performance as well.

4.4 Platforms

In this dissertation, one of the main goals is to provide the robot artificial intelligence to recognize vine trunks in a time-effective manner. So, the hardware platform used should provide high fre-

quency inference with low power costs. In this way, the Coral Egde TPU[1] was used as the main platform and provides high-performance ML inferencing for low-power devices. A comparison between the existing platforms is made, and Edge TPU is here briefly described.

4.4.1 Comparison between TPU, GPU and CPU

Wang et al. [50] propose a benchmark suite to compare three hardware platforms, Tensor Processing Unit (TPU), Graphical Processing Unit (GPU) and Central Processing Unit (CPU) for DL. In this study was used as hardware platforms the Google's Cloud TPU, NVIDIA V100 GPU and Intel Skylake CPU.

The main advantages and disadvantages of using TPUs that researchers presented in this paper are represented on table 4.1.

Advantages	Disadvantages
Have the highest training throughput, and are exceedingly optimized for CNNs.	GPU demonstrates better programming and flexibility for irregular computations
Speedup over GPU increases with larger CNNs.	CPU supports the largest models due to large memory capacity

Table 4.1: Advantages and disadvantages of using TPU.

4.4.2 Google Edge TPU

In 2018, Google announced the Coral Edge TPU[2] that provides to developers to have high-performance machine learning for low-power devices.

We can divide the development of machine learning into two stages. Initially, a model is trained with a large dataset on a powerful machine. Then the trained network is inserted into an application that needs to interpret real data. This "inference" stage is where Coral Edge TPU takes action, providing the capability to run these trained networks "at the edge" closest to the data. Thus, enabling the execution of deep-forward neural networks such as CNNs, making it ideal for a variety of vision-based ML applications.

The supported framework for Coral is TensorFlow Lite, which means that for a created TensorFlow model it is necessary to convert to a fully 8-bit quantized TensorFlow Lite model which is achieved by post-training quantization or quantization aware training.

[1] https://coral.ai/
[2] https://coral.ai/docs/edgetpu/faq/

4.5 Frameworks

The most popular frameworks for DL are TensorFlow, Keras, Caffe, PyTorch. TensorFlow is, by far the most widely used structure in the field of DL [25]. In this dissertation, TensorFlow[3] and TensorFlow Lite[4], were the programming frameworks used to create, train and test the models for the Edge TPU. In order to establish the interaction of DL models in robotics and real-time inference, ROS[5] is responsible for the communication between the different integrated systems and the natural interaction between the hardware and the developed code. All frameworks are here briefly introduced.

4.5.1 TensorFlow

TensorFlow is an open-source computing framework that uses data flow graphs, created by Google. This framework has become increasingly popular for training machine learning models, because of its high flexibility, rich algorithm library and support documentation, providing a good solution to solve real-world problems that require fast decision-making software.

4.5.2 TensorFlow Lite

TensorFlow Lite is a set of tools that came to help implement TensorFlow models on Internet of Things (IoT) and mobile devices, that consists of two essential elements:

- Interpreter: runs particularly optimized models in different hardware types.
- Converter: converts TensorFlow models in a simpler and more efficient format in order to be used by the interpreter.

Since this is designed to make it easier to perform ML on devices, it can also help to improve latency, privacy, connectivity and power consumption on-device.

4.5.3 ROS

ROS is an open-source framework for robotics and is commonly developed using C++ or Python programming languages. It works similarly to an operating system by providing tools and libraries that are easily implemented in real-time on robotic platforms. It provides hardware abstraction to a large variety of sensors, powerful simulators and easy communication between processes. Mainly, it works in a publisher-subscriber way to share messages over a network managed by a master.

[3] https://www.tensorflow.org
[4] https://www.tensorflow.org/lite
[5] https://www.ros.org/

4.6 Deep Learning Applications in Agriculture

DL applications have been increasingly implemented in several areas of agriculture. At the best of our knowledge, the detection of trunks in agriculture with DL, which is the main objective of this dissertation, is an area that has not yet been very well explored, until research date.

Among the various sectors of agriculture, Artificial Intelligence (AI) has made it possible to implement machines to perform tasks efficiently and effectively. Thus, computer vision algorithms based on DL techniques have been increasingly implemented in agriculture because it allows larger learning capabilities and thus, higher performance and precision [25]. In this way, DL in agriculture is a recent, modern and promising technique that is being used in the context of agriculture, such as fruit detection and counting, weed detection, plant disease detection, plant recognition and others [25].

To detect fruit in orchards, Bargoti et al. [51] create an accurate image-based fruit detection to support yield mapping and robotic harvesting. In this work, Faster-RCNN was used to detect mangoes, almonds and apples. The dataset is constituted with images of these three fruits and was increased using different data augmentation techniques resulting in a significant performance gain. However, the use of TL initialising the CNN directly from ImageNet feature showed no significant gains. This approach achieved an accuracy higher than 90% for apples and mangoes.

In the same context, to detect mango fruit Koirala et al. [28] made a new CNN architecture called MangoYOLO, based on characteristics of YOLOv3 and YOLOv2(tiny), to achieve better accuracy and speed. The training dataset is composed of more than 1300 tree images from five different orchards. The overall accuracy of the network running with a GPU is 98.3% with inference time of 8 ms per 512x512 pixel image and in other orchards images an accuracy of 89% with inference time of 70 ms per 2048x2048 pixel image (approximately 14 FPS).

Counting fruit with DL is also commonly used, for example, aiming to count tomato Rahnemoonfar et al. [52] propose a modified version of the Inception-ResNet architecture was used to detect and count the tomato fruit. The network was trained with generated synthetic data and tested on real data. The overall accuracy is 91% on real images and 93% on synthetic images.

Similarly, in [53] propose two CNNs with the objective of counting oranges and apples. The first CNN extracts different candidate regions in the images. The other is responsible for estimates the number of fruits in each region using a counting algorithm. The performance is analysed using oranges images collected throughout the morning and green apple images at dusk. The overall performance shows that better results are obtained with a limited dataset size and can perform on occluded fruits.

Fuentes et al. [54] propose a comparison for different detectors architectures, e.g., SSD, with different models, e.g., VGG net, in order to detect tomato plant diseases. In this work, Tomato Diseases and Pests Dataset was created, aiming to identify several infections and symptoms. Data augmentation techniques such as rotation, horizontal flipping, resizing and others were used in order to avoid overfitting. Finally, a comparison between data augmentation with and without is made, showing better results with data augmentation.

Likewise, Liu et al. [55] propose a fresh architecture based on AlexNet to detect apple leaf diseases. For this, a dataset with 13,689 images was created and used to train the network, in order to identify some usual apple leaf diseases. The result of this new approach shows an accuracy of 97.62%.

To identify and detect plant species Ghazi et al. [56] use TL to fine-tune pre-trained popular CNN architectures. The data was collected by using plant tasks datasets of LifeCLEF. In order to avoid overfitting and have more data to training, data augmentation operations such as rotation, translation, reflection and scaling, were applied to the original data. The system achieved an overall accuracy of 80%.

DL techniques are also used for image segmentation. Smith et al. [57] propose to segment roots in soil using U-Net. The time spent on labelling was extensive since the annotation needs to be manually and need to labelling all pixels considered to be root which takes for each image annotation approximately 30 min. Consequently, this work uses a dataset of 50 annotated images for training and 867 images for evaluation. Results show an overall accuracy of 70% and show that manual annotations are with less quality than the segmentation output.

This analysis of the literature leads us to the conclusion that DL techniques in agriculture are increasing exponentially. However, detecting trunks in vineyards using DL algorithms is nowadays a lack of state-of-the-art. Thus, this dissertation aims to overcome this gap with a low-power and high-performance DL-based to detect and segment vine trunks, able to perform for real-time operations in agriculture robotics.

Chapter 5

Implementation

In the implementation, various steps are delineated for the trunk detection, trunk segmentation and the creation of DL-based Assisted Labelling. Firstly, a dataset of vineyards trunks is created with the annotations for each image. Then, this dataset is used for the training of DL models for trunk detection and segmentation to be implemented in the Coral Edge TPU for real-time inference. The main steps are represented in Fig. 5.1.

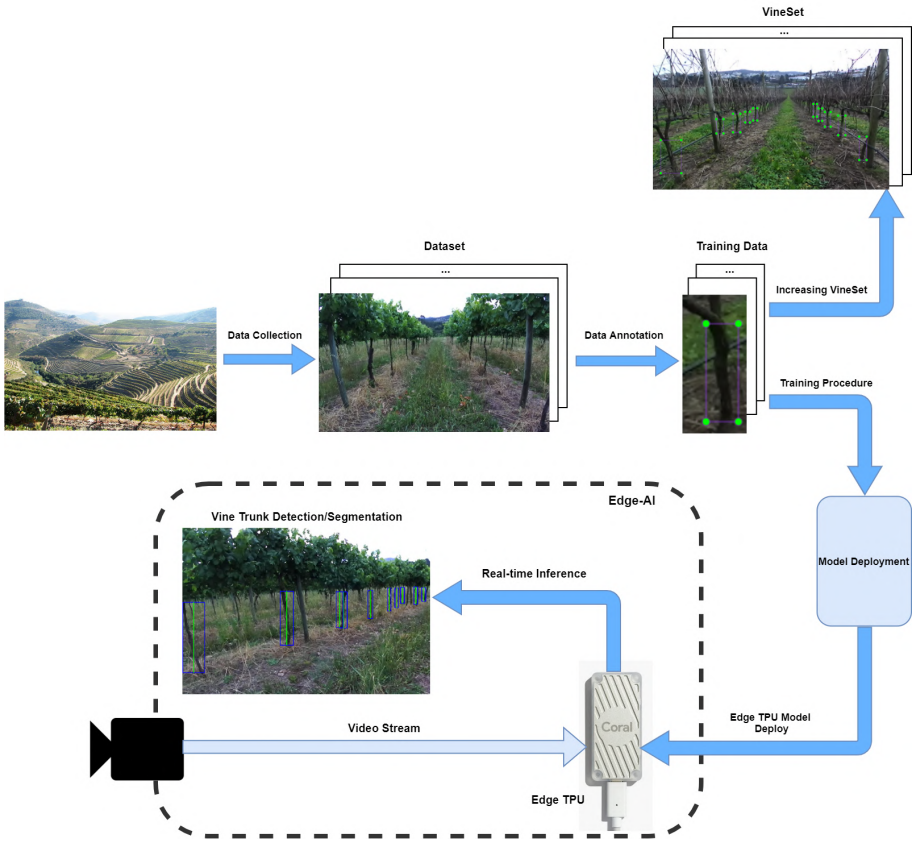


Figure 5.1: Vine trunk detection procedure flow.

5.1 VineSet

The detection of trunks in vineyards using DL algorithms is an area yet to be explored. Due to the non-existence of public datasets related to the detection of trunks in the vineyards, a dataset was created, called VineSet, an open-source dataset, was published in the Robotic Operating System (ROS) Agriculture community (http://wiki.ros.org/agriculture). This dataset was created using Agrob V16, a robotic platform for research and development of robotics technologies for Douro vineyards, that captured images containing vine trunks in different contexts and in different formats.

In the creation of VineSet, several steps were outlined. The first step went through by collecting images of 5 different vineyards, then annotations were made in all images and, finally, data augmentation was addressed to increase the dataset size.

5.1.1 Data Collection

In order to build the VineSet, our robotic platform, Agrob V16 collected several video streams in 5 different Douro Vineyards where 952 images were extracted from them, and some samples are presented in Fig. 5.2.

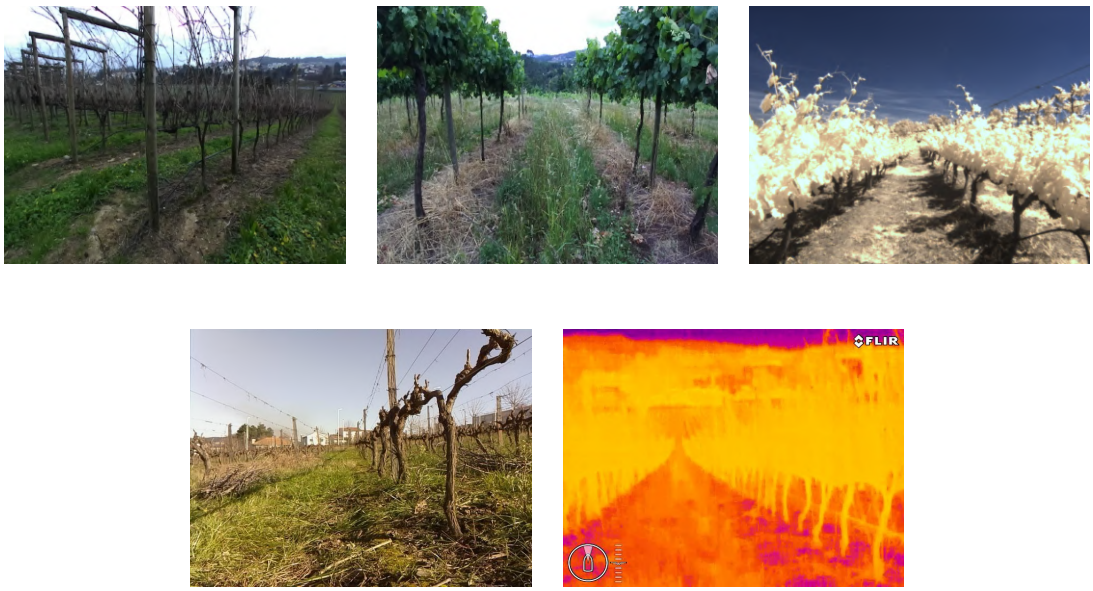


Figure 5.2: Set of different vineyards used in the VineSet data collection.

The data gathering was done under different stages of crop process and different times of the day, thus changing the lighting conditions. One of the most exclusive characteristics of VineSet is the fact that it contains thermal images, which are not included in many other datasets. This was possible since Agrob V16 is equipped with both an RGB and a thermal camera, which allows collecting a larger sample, thus increasing, even more, the diversity of images to the training

procedure. The advantage of using thermal images is the fact that they reduce the variability of images under different lighting conditions, thus allowing the detection of trunks to be more accurate.

5.1.2 Data annotation

From the data collection, trunks were manually annotated for all images. For this, LabelImg (https://github.com/tzutalin/labelImg) was the graphical image annotation tool used, which eases the process of labeling images with bounding boxes. The tool enables the user to quickly draw bounding boxes and annotate each box with a pre-defined label. These annotations are represented in a .xml file, using the PASCAL VOC annotation syntax, each of which represents a bounding box associated with the respective class and the four corners coordinates of each trunk. This process resulted in approximately 4600 trunks manually annotated and is also publicly available (http://wiki.ros.org/agriculture) together with the VineSet images.

In some cases, the vegetation in the vineyards causes obstructions in the identification of the trunks. Therefore, trunks that are not clear to distinguish with the background were not included in this annotation process. Thus, the trunks present in the vineyard corridor where the robot is located are manually annotated. The result of this process is represented in Fig. 5.3.

Figure 5.3: Result of annotation process for trunk detection.

This process aims to label the class and location coordinates of each bounding box containing the region of interest, which in this case are the trunks. In this way, we can have a ground truth that allows us to evaluate the prediction results during the various tests.

5.1.3 Data Augmentation

Even though DL outperforms most traditional ML methods in terms of precision and real-time application [32], one of the biggest challenges is to overcome overfitting. This is one of the frequent problems in ML that consists of modelling the data too well, learning only the expected output for each input instead of learning the general distribution of the input data.

Augementation operation	Description	Justification
Rotation	Rotation of the image 15, -15 and 45 degrees.	Makes the dataset more robust to possible irregularities of vineyards terrain.
Translation	Translate images by -30 to +30% on x- and y-axis independently.	Gives more possible situations to generalise the model avoiding overfitting.
Scale	Scale images to a value of 50 to 150% of their original size.	Gives more data to the training by zoom in or zoom out the image, giving a better train set.
Flipping	Mirror input images horizontally.	Flip images vertically make no sense since that hypothesis will never happen, so it has applied a flip horizontally that gives more data to the training.
Multiply	Multiply all pixels in an image with a random value sampled once per image. This augmenter can be used to make images lighter or darker.	Gives more situations of different situations of illumination due to the weather conditions.
Hue and Saturation	Increases or decreases hue and saturation by random values. The augmenter first transforms images to HSV colourspace, then adds random values to the H and S channels and afterwards converts back to RGB.	Gives more situations of different colours of the environment due to the different stations of the year.
Gaussian Noise	Add noise sampled from Gaussian distributions elementwise to images.	Gives to the robot the capability to be accurate when some noise affects the camera.

Table 5.1: Set of several augmentation operations used to expand VineSet.

(a) Translation (b) Multiply (c) Hue and Saturation

(d) Horizontal Flip (e) Rotation -15 degrees (f) Rotation 15 degrees

(g) Gaussian Noise (h) Combination of 3 operations

Figure 5.4: Set of several augmentation operations used to expand VineSet.

Furthermore, conditions such as variation of sunlight illumination during the day or the irregularities of slope vineyards may affect performance. In this way, to avoid overfitting and generalise the network, data augmentation is a usual method to enhance the variability of data for training by enlarging the dataset using label-preserving transformations.

In order to increase the diversity and robustness of the VineSet, the collected images were pre-processed with typical augmentation techniques used in DL works [37][54], resulting in an expansion of 8529 images. In this way the VineSet have a total of 9481 images. For this, imgaug (`https://imgaug.readthedocs.io/en/latest/`) was the library used for image augmentation. It supports a wide range of augmentation techniques such as horizontal flipping, image re-scaling, rotation, translation, multiply, Gaussian noise, hue and saturation. The augmentation operations performed are shown in Fig. 5.4 and described on table 5.1.

Besides, this library avoids the manual annotation process for the resulting images of augmentation, since not only augment images but also bounding boxes. So, if an image is rotated, the library can also rotate all bounding boxes on it correspondingly. This procedure allows to automatically generate the annotations for the augmented images as represented on Fig. 5.5.

(a) (b)

Figure 5.5: Result of an augmentation operation applying a 15 degree rotation on the (a) original image, resulting on the (b) augment image.

5.1.4 Training Procedure

In order to create a Coral Edge TPU compatible model suitable for real-time trunk detection, the training procedure needs to meet some requirements. The Edge TPU supports only TensorFlow Lite models that are fully 8-bit quantized and then explicitly compiled for it. In this way, it was used TensorFlow as a framework because it supports deployment in embedded and mobile devices with Tensorflow Lite enabling to convert Tensorflow models to the Lite version, as represented on Fig. 5.9.

Firstly, to verify the performance of different models with VineSet, our dataset, has been divided randomly into 90% training set and 10% testing set. These separated sets contain both the images and corresponding .xml files with the annotations. Then all the .xml files were converted

into two .csv files, one for the train set and other for the test set, that contains all the annotations for each trunk. This conversion is needed because TensorFlow only can convert .csv files into TFRecords. So, the next stage is to serialise the ground truth bounding boxes. This serialisation results in a TFRecord data type, which is a simple format for storing the data as a sequence of binary records. All this with the purpose of Tensorflow can interpret the data efficiently. These TFRecord files represent the input for the training of the different DL models. All of these steps are shown in Fig. 5.6. The next step is to train different CNNs. However, not all architectures can be used.

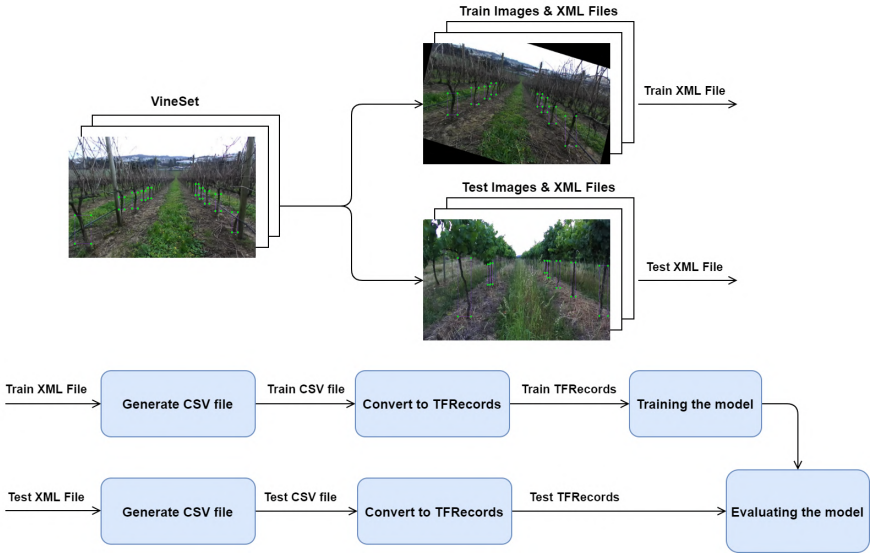


Figure 5.6: Training procedure flow.

5.1.4.1 Architectures and Models selection

The official TensorFlow Object Detection repository (`https://github.com/tensorflow/models`), contains the object detection architectures SSD and Faster R-CNN built on top of MobileNet and Inception and other models. However, the Edge TPU only supports a variety of operations and is typically compatible with models designed for mobile devices, that uses SSD architecture. Another fact is if a model is not able to be quantized with the quantization-aware training or full integer post-training quantization implies that is not compatible with Edge TPU. In [58], shows that Inception and MobileNets can be quantized. In this way, we choose SSD MobileNet V1, SSD MobileNet V2 and SSD Inception V2 that are available in TensorFlow object detection library which meets the above requirements and is pre-trained on the COCO dataset [6]. Thereby, it is possible to use TL in order to achieve the objective faster.

5.1.4.2 Hyperparameter selection

The task of choosing the hyperparameters is a crucial step to improve the performance. The models chosen in section 5.1.4.1 had already hyperparameters values set, that had been analysed

in order to reach the optimal performance for each model. These values are used as default in TL. However, some of them should be modified. In this way, to modify the hyperparameters requires editing the model configuration file that is available for each model.

The batch size is one of the hyperparameters. It represents the total number of training samples present in a single batch, and smaller batch size implies less memory and faster training. However, the smaller the batch, the less accurate. So, taking into account memory and time constraints, hyperparameters have been adjusted. Thus, the batch size was set to 18. In order to obtain the number of iterations required for the training, several tests were performed to observe when the training error curve began to converge. For TL the number of training steps was 50,000, which is approximately 95 epochs. For models trained from scratch the number of training steps was increased to 100,000, which is approximately 185 epochs, since all parameters are initialise from zero. Besides, all the images were resized to a resolution of 300x300 by the `image_resizer` hyperparameter. The width and height of the resized image to the input can be chosen. Parameters such as momentum, learning rate, resolution and others, have not been changed, being preserved the values that came with the original model.

5.1.4.3 Training

During the training of DL models, was noticed that different GPUs have different times per iteration. So, better GPUs leads to faster training. Along with this, not every GPU can handle with large batch sizes since it exceeds the available memory. To solve this problem, the training procedure was performed in an online platform, Google Colaboratory (https://colab.research.google.com). This is a browser-based platform that allows us to train our models on machines with an NVIDIA Tesla P100-PCIE for free, avoiding the restriction of reduced computational power and produces results much faster when compared to the available hardware. However, the fact being free has some limitation for long-running tasks. So, Colab provides a maximum GPU runtime of 8 to 12 hours, ideally at a time. Though, this was not a problem, since it is possible to retrain the model from the point where it previously ended.

Initially, the training was done using TL. So, fine-tuning consists of using the pre-trained network to initialise all the weights. In this way, in order to train from a pre-trained model, i.e., using TL, it is necessary to edit the parameter `train_config` in the model configuration file adding a fine-tune checkpoint. Fine-tuning can be done in two ways. The first restore all variables in the graph that are available in the model binary file, and the box predictor (last layer) weights are also restored. The second way restores just the weights from feature extractor, i.e., all the layers in backbone networks, like MobileNet. Therefore, if the parameter `fine_tune_checkpoint_type=detection`, then it restore all variables. In the other hand, `fine_tune_checkpoint_type=classification` then it restore only the variables from feature extractor. Since our dataset was reasonably large is recommended restoring all weights from the checkpoint which implies that `fine_tune_checkpoint_type=detection`. So, the checkpoint was used to initialise all the weights and other variables. During the training, all

layers weights are adjusted, because unfrozen layers give better results than frozen. Train with TL reaches faster the objective. Thus, the training steps was set to 50k.

Also, experiments were carried out training the models from scratch to analyse, which would reach faster a satisfactory result. In this case, was necessary to eliminate from the parameter `train_config` in model configuraton file the fine-tune checkpoint and the parameter that was used before, the `fine_tune_checkpoint_type=detection`. In this way, the weights are randomly initialised and adjusted during the train. Training from scratch implies more iterations than fine-tune, and more time. For this, it was increased to 100k the training steps in order to get the same or better result as TL.

In both cases, training with TL and from scratch, it was used the quantization-aware training by adding to the model configuration file the `quantization` parameter, that simulates the impact of 8-bit numbers in training using quantized nodes in the neural network graph.

So, the next step was to train the differents DL models with these two different approaches, together with VineSet. In this manner, to train is only needed to use the training script from TensorFlow repository (`https://github.com/tensorflow/models`), upload TFRecords files, download the folder of the model chosen with the respective checkpoint and pipeline file, and finally run the script.

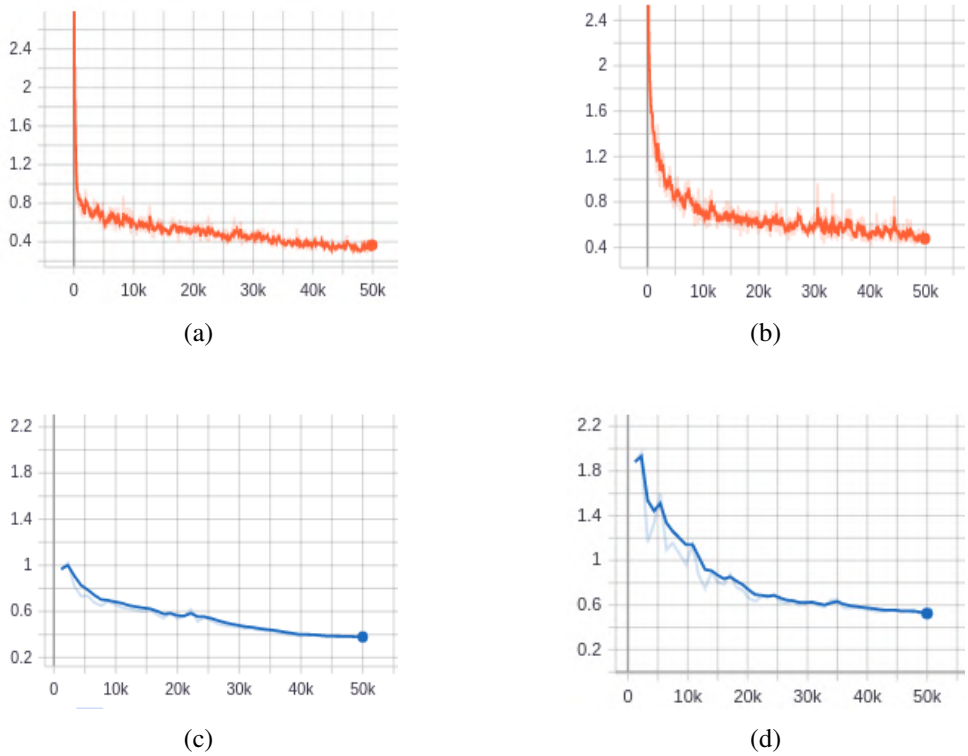


Figure 5.7: Loss result of 50k iterations. Train loss (a) using transfer learning and (b) from scratch. Validation loss (c) using transfer learning and (d) from scratch.

At the end of the training, the generalisation of our model was analysed from Fig. 5.7. It is possible to see that the training loss decreases in the same way as validation loss decreases

in both cases, using TL and from scratch. In the case of overfitting, the training loss decreases while validation loss increases. Thus, our model does not suffer from overfitting, proving that our dataset, VineSet, is robust. Besides, train loss converges faster in TL than from scratch, proving that it is necessary to have more iterations in training from scratch.

After training and checking the model, we move on to creating a model that's compatible with the Edge TPU. Thus, a Tensorflow model is generated that can be later retrained or used to train another's models by using TL. However, this model is exported to a frozen graph. Then, the frozen graph is converted into a TensorFlow Lite model that is entirely 8-bit quantized. Finally, the Edge TPU compiler divides the different inference operations that run on the Edge TPU and the CPU, as illustrated in Fig. 5.8.

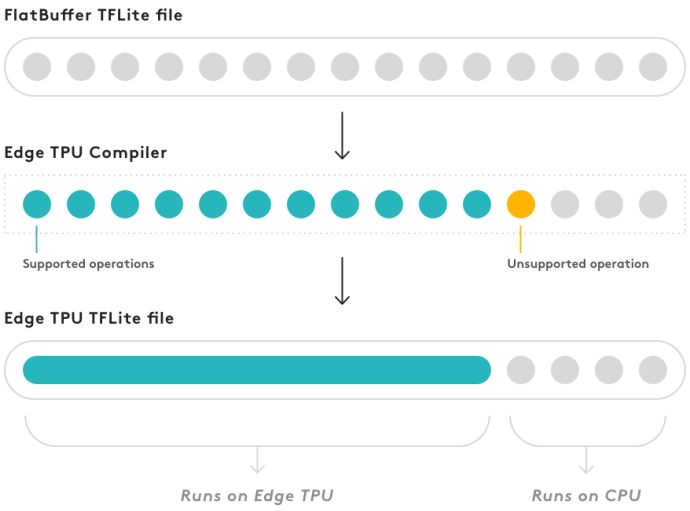


Figure 5.8: Edge TPU model compilation scheme [17].

Table 5.2 shows the supported and unsupported executed by SSD MobileNet V1 on the Edge TPU after the compilation. The operations that have the status of mapped to Edge TPU are the ones that are supported operations and execute on the Edge TPU, everything else that is unsupported is compiled into a custom operation that executes on the CPU. So, the more number operations assigned to the Edge TPU the faster the inference is.

Operator	Count	Status
Depthwise 2D Convolution	34	Mapped to Edge TPU
Reshape	13	Mapped to Edge TPU
Logistic	1	Mapped to Edge TPU
Concatenation	2	Mapped to Edge TPU
2D Convolution	13	Mapped to Edge TPU
Custom	1	Unsupported data type

Table 5.2: Operation log output of Edge TPU compiler for SSD MobileNet V1.

At this point, a Edge TPU model is deployed to the Coral Edge TPU. All this process is represented on Fig. 5.9.

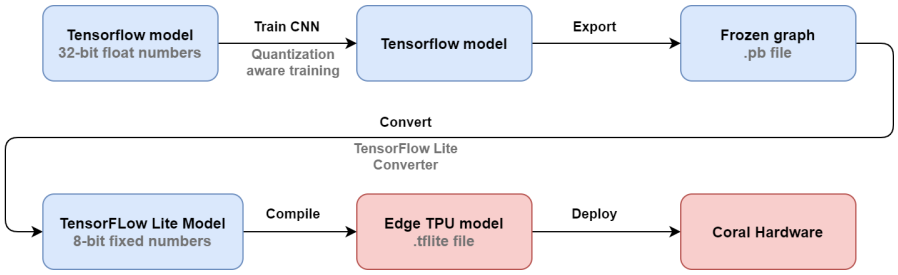


Figure 5.9: Training procedure flow.

5.1.5 Real-time inference

In order to test our models, an environment that simulates real-time inference is needed. For this, ROS nodes were used together with Coral. This environment had been built before and is available in the CRIIS repository (`https://gitlab.inesctec.pt/agrob/agrob_vineset`). This consists of two packages, the `edgetpu_cpp` that contains the Edge TPU API for object detection, and the `detector` that contains the ROS node to call the API on subscribed images. So, to run a real-time inference with Coral is required to set up the `detector` node parameters. There are three parameters: the `image_topic`, i.e., the name of the image topic to subscribe; the `model_path`, i.e., the path to the model `.tflite` to use; and the `labels_path`, i.e., the path to the label file.

After everything set up, the Coral Edge TPU needs to be plugged-in on the computer where the ROS node is executed. Then, real-time inference occurs using the frames of a video stream. In order to obtain this inference time to be later used as an evaluation metric for each DL model, a clock in the `std` library already implemented earlier was used. In this way, the time between the detector returning the resulting bounding boxes and the image reception is obtained. Fig. 5.10 shows the result of real-time inference for trunk detection.

Figure 5.10: Real-time inference using Edge TPU for trunk detection.

5.1.6 Measuring and evaluating models performance

In order to compare the performance of state-of-the-art DL object detection models for trunk detection on VineSet dataset, it was necessary to have an evaluation method in order to conclude. So, the performance of our system is evaluated using the Average Precision (AP) based on intersection-over-union (IOU) introduced in the PASCAL Visual Object Classes (VOC) Challenge [24] and the F1-Scores (F1) that is another popular metric used to evaluate object detection algorithms [25]. In order to run these metrics, a `Python` program was used that is available to the public at (`https://github.com/rafaelpadilla/Object-Detection-Metrics`). This program allows us to measure the AP for each model trained. Additionally, the F1 metric was added to the file by just getting the precision and recall already calculated and use the equation 3.10. Also, the speed of the DL model, together with VineSet in the detection performance, was explored. Thus, the indicator chosen to evaluate the performance of the speed of each DL model was the inference time, calculated in section 5.1.5.

Therefore, the input for this program is a `.txt` file for both ground truth and detection containing the coordinates of the bounding boxes. In this way, a `C++` program was used. This program uses the Edge TPU API and runs the DL model trained to make the inference on the test set. This test set composed of 641 images, with approximately 2000 trunks, was randomly extracted from the dataset before training from a total of 9481 images. Then `.txt` files were generated with the corresponding detection bounding boxes and confidences for each image. Then, IoU threshold was set to 0.5 in order to compare the estimated results with the ground-truth.

After this, a `Python` script runs and compares all the corresponding bounding boxes between the ground truth and the detection and gives as output the result of the above-mentioned metrics.

5.2 VineSet for Trunk Segmentation

In order for the robot to have a better notion of the trunk structure and provide better input for locating and mapping, the trunk segmentation was proposed. This task was performed in precisely the same way as trunk detection. However, in this procedure, the purpose was not to detect the trunk as it was previously done. Instead, the goal is to detect the various pieces of the trunk in order to segment it. Therefore, object detection models were used together with VineSet to segment the trunk. It is worth noting that this is a practical way of emulating the pixel-based segmentation performed by, for example, semantic segmentation models.

5.2.1 Data annotation

The data annotation previously performed do not serve for this procedure because this time, the objective is to segment the trunk and not just detect it. In this way, it was necessary to make the annotations again from scratch. As each annotation represents a bounding box, in order to segment the trunks, it was necessary to place many bounding boxes to fill the trunk structure, as represented on Fig. 5.11. In the annotation process, if it was not possible to clearly distinguish whether or not

Figure 5.11: Result of annotation process for trunk segmentation.

it was part of the trunk, then it was not annotated. In order not to affect the training, all these details were taken into account by labelling only those parts that are really part of the trunk. So, a new set of annotations was added to all the VineSet images, providing the user the ability to train models for segment vine trunks.

5.2.2 Data Augmentation

In the same way as section 5.1.3, the same several data augmentation operations were applied in order to make the dataset more diverse and robust. Fig 5.12 shows the result of data augmentation for trunk segmentation.

(a) (b)

Figure 5.12: Result of an augmentation operation applying a -15 degree rotation on the (a) original image, resulting on the (b) augment image.

5.2.3 Modification in the model SSD MobileNet V1

This entire segmentation process involves smaller bounding boxes so that the trunk structure is filled. As SSD architectures are not the best for the detection of small objects as evidenced in the

original paper [13], a modification has been made to the SSD MobileNet V1 model in order to see if the result improves.

The SSD architecture uses a base network, in this case, MobileNet V1 as a feature extractor, then the SSD adds six more auxiliary layers for object detection. However optionally base network layers can be used, and that is what happens in our case. It is from these six feature maps that the predictions are made. The lower the layer, the higher is the resolution of the feature map. So, feature maps with a higher resolution can detect smaller objects, and feature maps with low resolution detect larger objects. Besides, all the layers that have an arrow for the detections on the Fig. 5.13, higher layers, are responsible to make predictions. On the other hand, lower layers are responsible for the detection of low-level features like edges and colours.

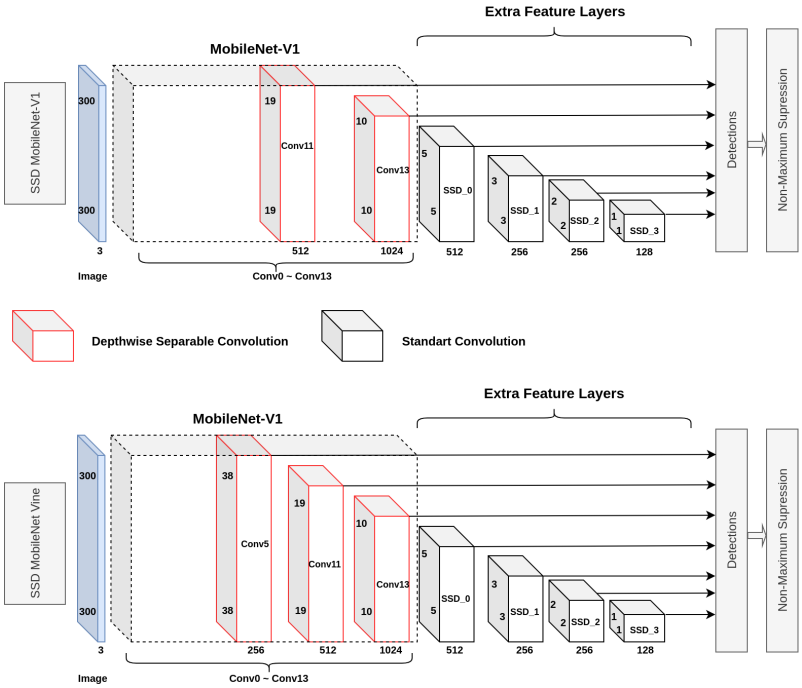


Figure 5.13: SSD MobileNet V1 and SSD MobileNet V1 modified by adding a feature map for detections with a higher resolution (38x38).

So, layers have different feature maps sizes, and they depend on the resolution of the input image. With an input image resolution of 300x300, the layers 4 to 5 have a 38x38 feature map size, the layers 6 to 11 have a 19x19 feature map size and layers 12 to 13 have a 10x10 feature map size. In Fig. 5.13 the original SSD MobileNet-V1 only uses the layer 11 and 13 to make detections. So, in order to improve the performance of SSD-MobileNet V1, in this work is proposed the addition of a layer to the feature extractor, Conv5. This layer is a feature map with a resolution of 38x38, higher than the original ones. To perform this creation, the original model was changed using the available model configuration file from the TensorFlow framework. The name of this model modified was set to SSD MobileNet-Vine in order to be easy to compare the results with the original model.

5.2.4 Training Procedure

The training procedure to trunk segmentation with object detection models was the same that was used on section 5.1.4. All the hyperparameters for training are equal too. However, as it concerns the detection of smaller bounding boxes, some recommendations described in the SSD architecture article [13] were used in order to improve the detection of smaller objects. This way, two different training configurations were used for each model. One train was with an input size of 300x300. The other train with a 512x512 resolution in order to improve the final result. Along with this, the model modified in section 5.2.3 was trained with both resolutions too.

Also, this work proposes the training of another architecture to see if it improves the detection of small objects. So, looking at the pre-trained architectures available at TensorFlow Object Detection repository, it only rests the Faster-RCNN, so that was the chosen from state-of-the-art DL architectures. Although not compatible with Edge TPU, the inference procedure was made using a Colab GPU. This architecture is built on top of ResNet50 model [59]. All the hyperparameters have not been changed, remaining the same as the original model. Since the batch size is 1 for this architecture the number of iterations is necessary more than the used for SSD architectures.

5.2.5 Real-time inference with segmentation and detection simultaneously

After training all the object detection models for segmentation of trunks and compiling all the models that have SSD architecture using Edge TPU compiler, the detector node parameters in section 5.1.5 were duplicated in order to introduce the segmentation model. In this way, the node can publish the detections of the images subscribed by using both models, one for trunk detection and the other for trunk segmentation. Thus, the Edge TPU not only runs the model to detect the trunks, as also runs simultaneously the model to segment the trunks. Fig. 5.14 shows the result of real-time inference for trunk detection and segmentation simultaneously.

Figure 5.14: Real-time inference using Edge TPU for trunk detection and segmentation simultaneously.

5.2.6 Measuring and evaluating segmentation performance

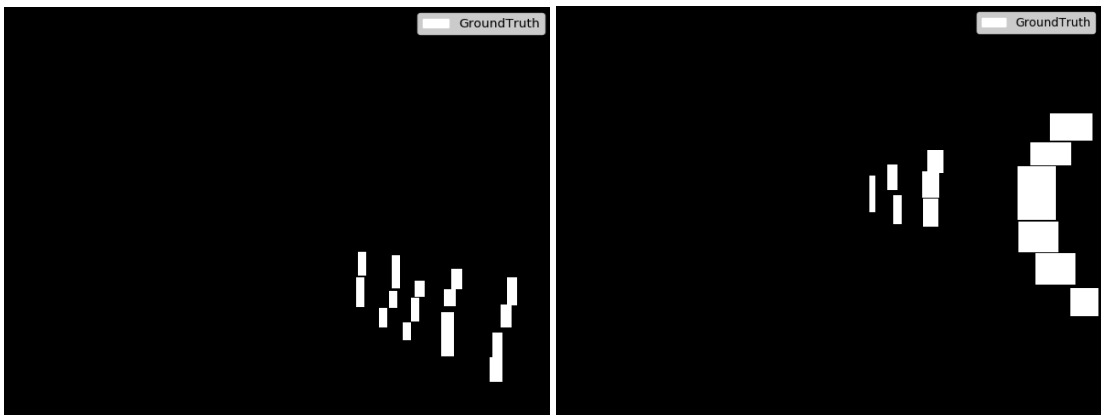
The metrics used previously in section 5.1.6 to detect the trunks are not suitable to evaluate the segmentation of the trunks because they do not give us a precise result regarding the segmentation of the trunk.

In order to measure and evaluate the performance of state-of-the-art DL object detection models for trunk segmentation on VineSet dataset, a `Python` program was built and is available at CRIIS repository (`https://gitlab.inesctec.pt/agrob/agrob_vineset`). The propose of this program is to evaluate the several models using the Jaccard Similarity Index (JSI) and the Dice Similarity Index (DSI), that are the most popular metrics used to evaluate semantic segmentation algorithms [29][30]. Also, the inference time was explored, as in section 5.1.6.

The first step in the creation of this program was to read the two .txt files generated in the same way as the section 5.1.6, for ground truth and for detection, that contains the coordinates of the bounding boxes. Since the metrics verify if the pixel belongs to a certain class and does not use bounding boxes, it was necessary to create binary images for both ground truth and detections. This way, we started by creating a whole black image, i.e., with all pixels with value 0. Then, all bounding boxes of both the ground truth and the prediction boxes were checked, and the values of the pixels were modified within the limits of these bounding boxes with the value 255, i.e., by white pixels, resulting on a binary image for ground truth represented on Fig. 5.15a and Fig. 5.15b , and for prediction represented on Fig. 5.16a and Fig. 5.16b.

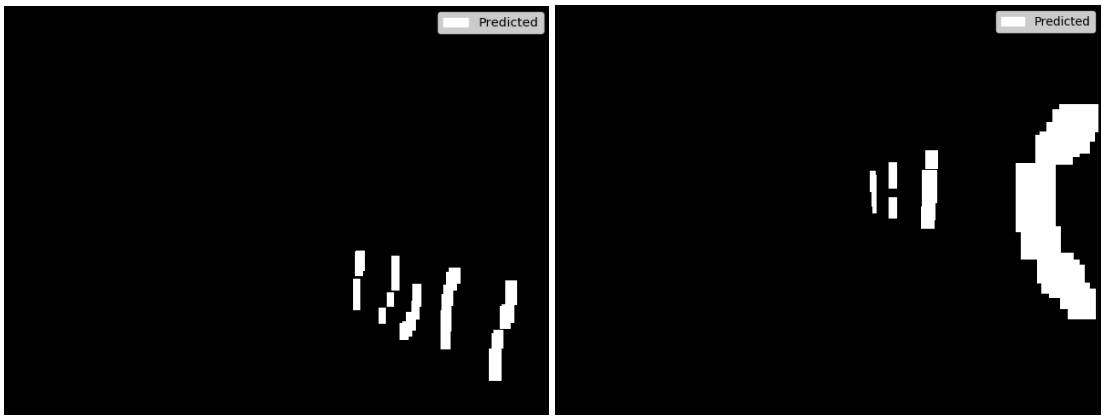
	Operation
TP	G AND P
TN	(NOT G) AND (NOT P)
FP	(NOT G) AND P
FN	G AND (NOT P)

Table 5.3: Binary operations applied to get TP, TN, FP and FN.



(a) (b)

Figure 5.15: Set of 2 binary images of ground truth.



(a) (b)

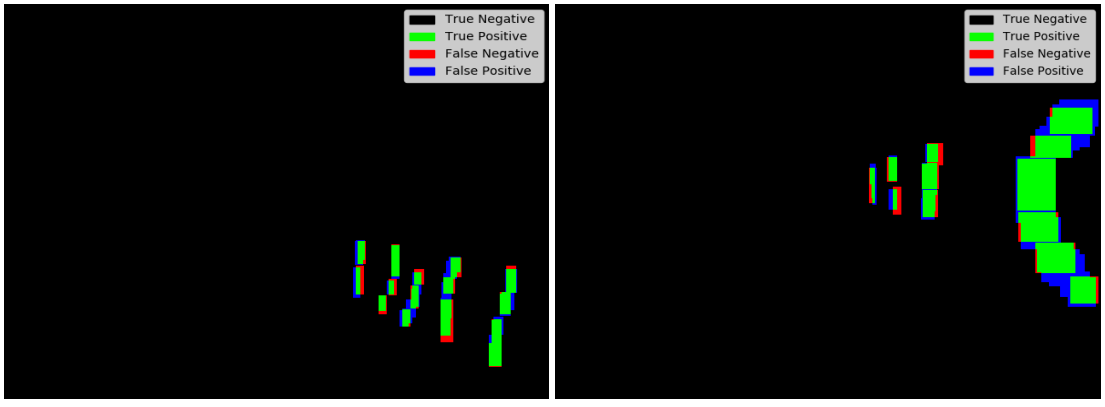
Figure 5.16: Set of 2 binary images of prediction.

From the moment we have these binary images, it was possible to perform several binary operations. These operations were done in order to obtain True Positives, True Negatives, False Positives and False Negatives, as represented on table 5.3, where G is the ground truth binary image and P is the binary prediction image.

From these operations, the TP, TN, FP and FN were used to calculate the metrics to evaluate the performance of our models.

To do so, JSI and DSI can be calculated in two ways. The first is as described on the equation 3.7for JSI and equation 3.13 for DSI, where is used the TP, FP and FN. The second way is by using the areas as described on the equation 3.5 for JSI and equation 3.11 for DSI, where all the white pixels were counted on both ground truth and prediction binary images and used as the areas.

Also, in order to make the TP, TN, FP and FN more perceptible, an RGB image was created, as represented on Fig. 5.17a and Fig. 5.17b, where the TPs were marked in green, the TN in black, the FP in blue and the FN in red.



(a) (b)

Figure 5.17: Set of 2 images with TP, TN, FP and FN.

(a) (b)

Figure 5.18: TP represented with white pixels on original images.

To see the pixels that were detected and that are in accordance with the ground truth, i.e. the TP pixels, in the original image, all the pixels from TP binary image that are white were passed to the original image, resulting on Fig. 5.18a and Fig. 5.18b.

5.3 VineSet for Semantic Segmentation

In this work, a brief introduction to the use of semantic segmentation models for trunk segmentation was finally made. In this way, the repository (https://github.com/GeorgeSeif/Semantic-Segmentation-Suite) is a way to train semantic segmentation models through TensorFlow like FC-DenseNet103 [60] and DeepLabv3 [61]. These models, unlike the ones seen above, generate as output a mask directly. Then, each pixel is classified to a particular class. Thus it is pixel-level image classification.

To do this train, Colab was used once again. The first step in training our segmentation model is to prepare the dataset. To do so, VineSet was used and has been divided randomly into 90% training set and 10% testing set. The training requires as input RGB images and the corresponding ground truth segmentation masks. These masks were the ground truth binary images obtained in section 5.2.6. Thus, these images were placed in separated folders in the training/testing folder.

However, unlike object detection models that use .xml files that take up less space for their annotations, the semantic segmentation models for each image require a ground truth segmentation mask implying a larger file size. Thus, the upload of the train/test set images from VineSet was limited. If all VineSet images were used for training, the training input would be huge and the training time would increase dramatically. As this is a preliminary approach, it was chosen to use only a portion of the VineSet.

Then, the hyperparameters are maintained as the original ones, the batch size is 1, and the number of iterations was 40 epochs which is approximately 138,000 iterations, for both models trained from scratch. Besides, during the training, semantic segmentation models take an input image and break it into several crops of the same size. Each crop fed the CNN to get the classified

category for that crop as an output. This crop size is defined initially and needs to be divisible by 2^n, where n is the number of pooling layers. Since FC-DenseNet103 has five pooling layers and DeepLabv3 has four pooling layers, the crop size chosen was 480x480.

5.3.1 Measuring and evaluating performance

In order to measure the performance of these models, the Python metrics program created in section 5.2.6 was used. Since our program utilises the most used metrics in semantic segmentation,i.e., JSI and DSI, it was possible to evaluate all images of predictions images generated from the test set. These prediction images were obtained through the inference process performed from a prediction script available in the repository mentioned above, using a Colab GPU.

5.4 Deep Learning-based Assisted Labelling

Training a DL model involves several steps, one of the most important of which is data annotation. Generally, this step is a long process, and the time spent depends on several factors, such as the total number of images that the dataset has, the number of classes and the ease of manually identifying the bounding box corresponding to each class. Thus, this dissertation proposes to create an assisted labelling procedure that uses AI to help the annotation process in the detection of trunks in the vineyards.

5.4.1 Functionalities and Interface

In this way, a python notebook is made available in the CRIIS repository (`https://gitlab.inesctec.pt/agrob/agrob_vineset`), which is open to the general community. The procedure of this new solution consists of using an online platform, Google Colaboratory (`https://colab.research.google.com`), so that the user can save the resources of his machine. This python notebook is an adaptation of a tutorial available at TensorFlow repository (`https://github.com/tensorflow/models`) that only allows us to see the inference result.

This tool provides a DL model for detecting vine trunks, trained in this dissertation with VineSet, capable of detecting trunks in other contexts such as orchards or forests. So, an essential factor for automating this process is the use of the DL model. Taking into account the results obtained on section 6.1, the SSD MobileNet-V1 trained with VineSet was the model chosen for the detection of trunks in the images introduced in this tool. For that, the frozen graph file exported in section 5.1.4.3 was used here to run the inference. The assisted labelling procedure uses this model to pre-process the user dataset, automatically annotating the detected trunks and saving the annotations in a `.xml` file with the Pascal VOC format for each image. The user can then load the automatic annotations and complete them manually.

This entire process is summarized in the information and instructions, as represented in Fig. 5.19 that the user encounters when using our tool to have a simple and effective experience. All the steps that the user needs to perform are separated by different topics making it easier to understand

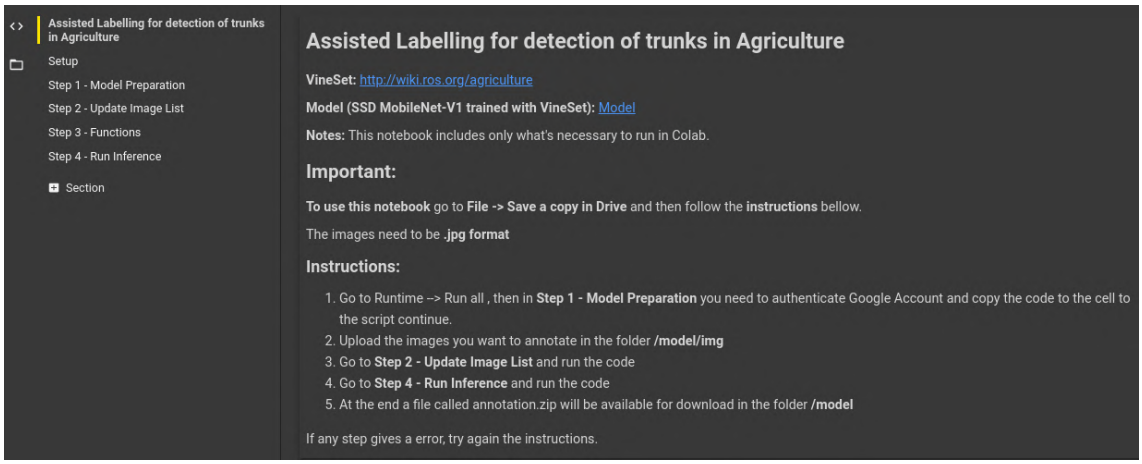


Figure 5.19: Information and instructions of our Assited Labelling interface.

the interface, as represented in Fig. 5.20. Summarizing, this tool, in addition to using the trained DL model together with VineSet, is a way to expand VineSet faster and iteratively improve DL-based object detection models performance. In this way, this tool provides feedback in the training of DL models.

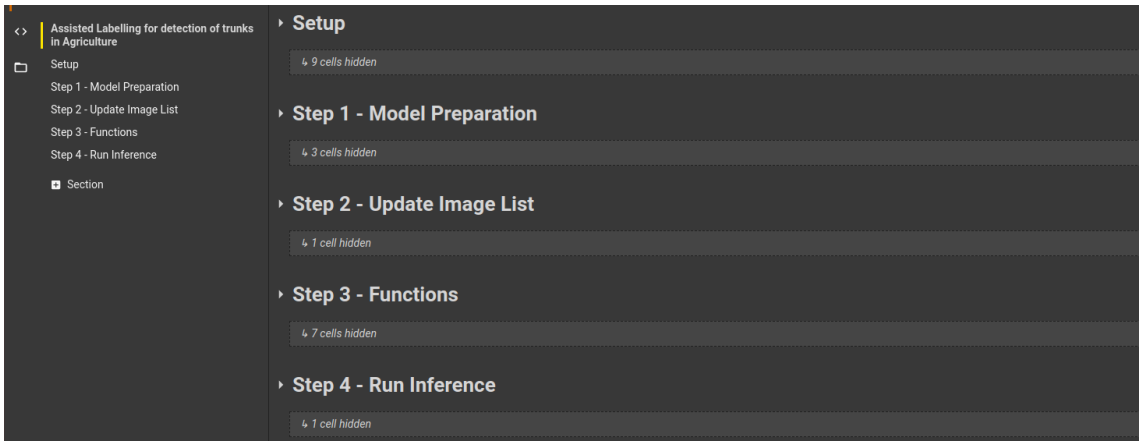


Figure 5.20: Main steps of our Assited Labelling interface.

5.4.2 Measuring and evaluating performance

To assess the performance of our assisted labelling procedure tests were carried out in which several factors were analyzed in comparison with manual annotation. In order to make conclusions, the average time to manually label a trunk was measured by the total time spent on noting the trunks in 20 different VineSet images, divided by the total number of trunks in these images, resulting in an average of 5 seconds per trunk. Thus, once this value is established, it will be essential to calculate the total time spent on several images. For this, the percentage of annotations made

automatically and the percentage of annotations made manually were calculated. In this way, the total time spent with our tool is calculated by the equation 5.1.

$$t_{assisted} + TOTAL_{trunks} \times \%_{manual} \times 5 \tag{5.1}$$

Where $t_{assisted}$ is the time spent by our automatic annotation tool, $TOTAL_{trunks}$ the number of total trunks and $\%_{manual}$ the percentage of manual annotations.

In order to assess the scope of our tool in other areas of agriculture using VineSet, we organize a range of images. These images are from different orchards and forests, where we evaluate the performance of our assisted labelling compared to the manual one.

Chapter 6

Results and Discussion

In this chapter, the results for the performed work are shown and then analyzed in the discussion. Firstly, the DL object detection models are evaluated to compare and analyze the performance in real-time of trunk detection. Then, the same models used before and a modified model are compared with different input resolutions (300x300 and 512x512) and evaluated for segmenting trunks. Additionally, an architecture not supported by Coral Edge TPU is evaluated for the segmentation of trunks. Also, semantic segmentation is here introduced. Finally, our Assisted Labelling Procedure is evaluated and compared with some state-of-the-art tools.

6.1 VineSet Trunk Detection Results

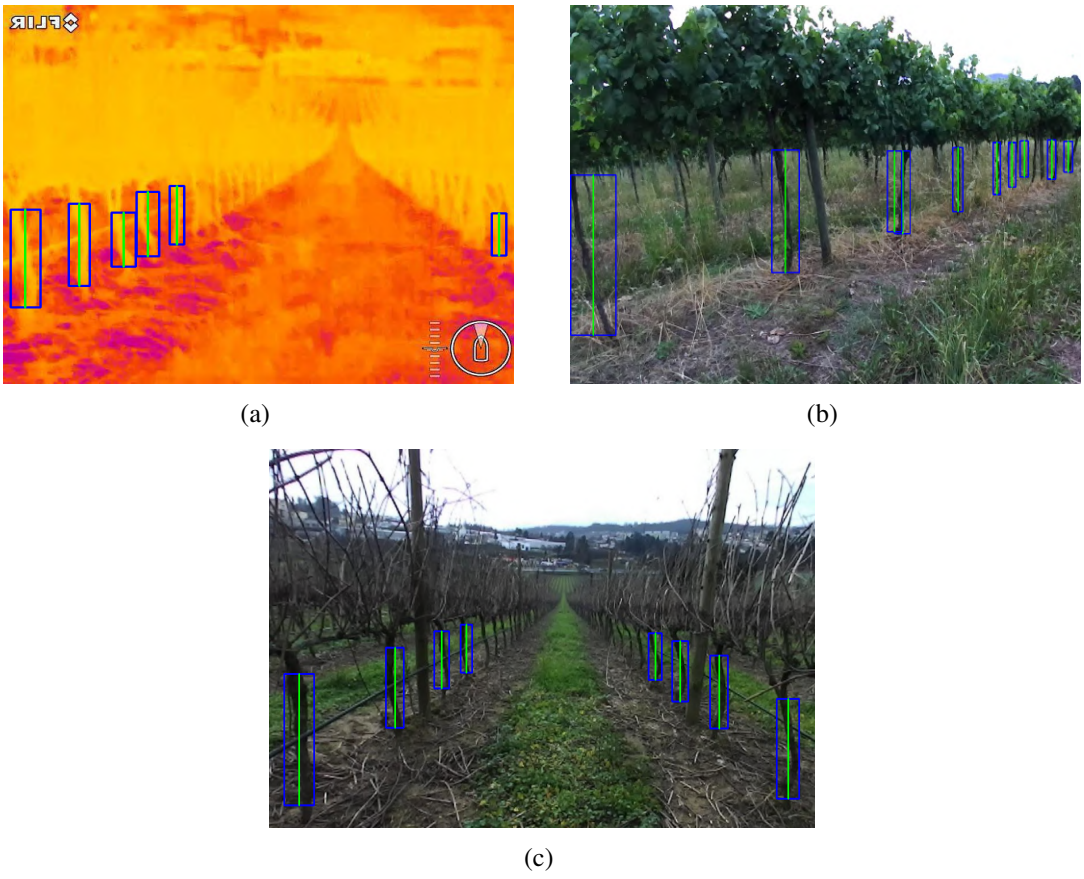
Model	Resolution	Inference time (ms)	Fine-tuning		From scratch			
			50k		50k		100k	
			AP(%)	F1	AP(%)	F1	AP(%)	F1
SSD MobileNet-V1	300 x 300	4.55	84.16	0.841	68.44	0.685	85.93	0.834
SSD MobileNet-V2	300 x 300	5.04	83.01	0.808	60.44	0.639	83.70	0.812
SSD Inception-V2	300 x 300	23.82	75.78	0.848	58.05	0.658	76.77	0.849
SSD MobileNet-Vine	300 x 300	6.22	-	-	32.54	0.448	65.40	0.635

Table 6.1: AP (%), F1 Scores and average inference time per image (ms) using Coral Edge TPU, with fine-tuning and from scratch training.

In order to evaluate DL models for trunks detection on top of Google's Edge TPU the performance of the three models, i.e., SSD MobileNet-V1, SSD MobileNet-V2 and SSD Inception-V2 is compared using all the evaluation data composed by 640 images with approximately 2000 vine trunks. Also, the model for object detection created to the segmentation process, i.e., the SSD MobileNet-Vine, is equally compared with the other models. Thus, the global performance of the different detectors is evaluated. For this, as referenced in section 5.1.6, the AP and F1-Scores metrics were used with an IOU threshold defined to 0.5. Also, the speed of the models, together with

Edge TPU in the detection performance, was explored. Thus, the inference time is also evaluated. Table 6.1 summarizes the AP, F1-Score and run time performance for all models.

Fig. 6.1 shows an example of detection for three different vineyards, provided by SSD MobileNet-V1, that achieves the best result for AP metric and inference time.



(a) (b)

(c)

Figure 6.1: Detection results using SSD MobileNet-V1

6.1.1 Discussion

The comparison on table 6.1 for training with fine-tuning shows that the SSD MobileNet-V1 model obtained the best AP result of 84.16%. Although the SSD MobileNet-V2 has a slightly lower AP result than the SSD MobileNet-V1, they are very similar, which was expected, since they have a very similar architecture. Also, they obtained the best time of inference, because Mobilenet uses depthwise separable convolution, while Inception uses standard convolution, resulting in fewer parameters on MobileNet compared to Inception V2. However, sometimes this results in a slight decrease in performance as well, which in this case was not verified. Also, the inference time was lower due to the high-performance ML inferencing of Edge TPU, which allowed to perform in real-time. For example, an average inference time per image of 4.55 ms was obtained for SSD MobileNet-V1, which results on a speed of 220 FPS, making this model ideal for real-time performance. Considering that this work uses lightweight models oriented to embedded devices,

the AP results revealed to be entirely satisfactory on the object detection procedure. In this way, were obtained reliable detectors, suitable for execution on autonomous robotic platforms due to low energy consumption. F1 scores show us that SSD Inception-V2 obtains the best result. This conclusion is since it has higher precision and a slightly lower recall compared to other models.

Regarding training from scratch, the number of steps had to be doubled to achieve similar or better results than fine-tuning. With only 50,000 steps, the performance obtained is significantly lower, as shown in table 6.1. This approach leads to the conclusion that, when training from scratch, the model needs to learn low and high-level semantics, so more iterations are needed to converge correctly. Thus, more effort is required in order to achieve satisfactory results [62].

For the detection of the trunks, the modified model of the SSD MobileNet-V1 was also tested, in this case, we gave a different name, SSD MobileNet-Vine, to be able to identify and compare with the original model. The change added a higher resolution detection feature map that would possibly improve the detection of smaller objects for trunk segmentation. However, as we can see in the table 6.1, this modification do not improve the result, which was expected since we are detecting trunks up to 3 meters away which compared to the image is still a large object.

In conclusion, this presents a solution to the non-existence of state-of-the-art works to detects vine trunks on images using DL models on the Edge TPU that is yet not popular in the literature. Also, the results obtained for SSD MobileNet-V1 shows that our dataset, VineSet, gives good results for trunk detection and can be implemented in real-time applications. Thus, the detections can be used both on mapping and localization task.

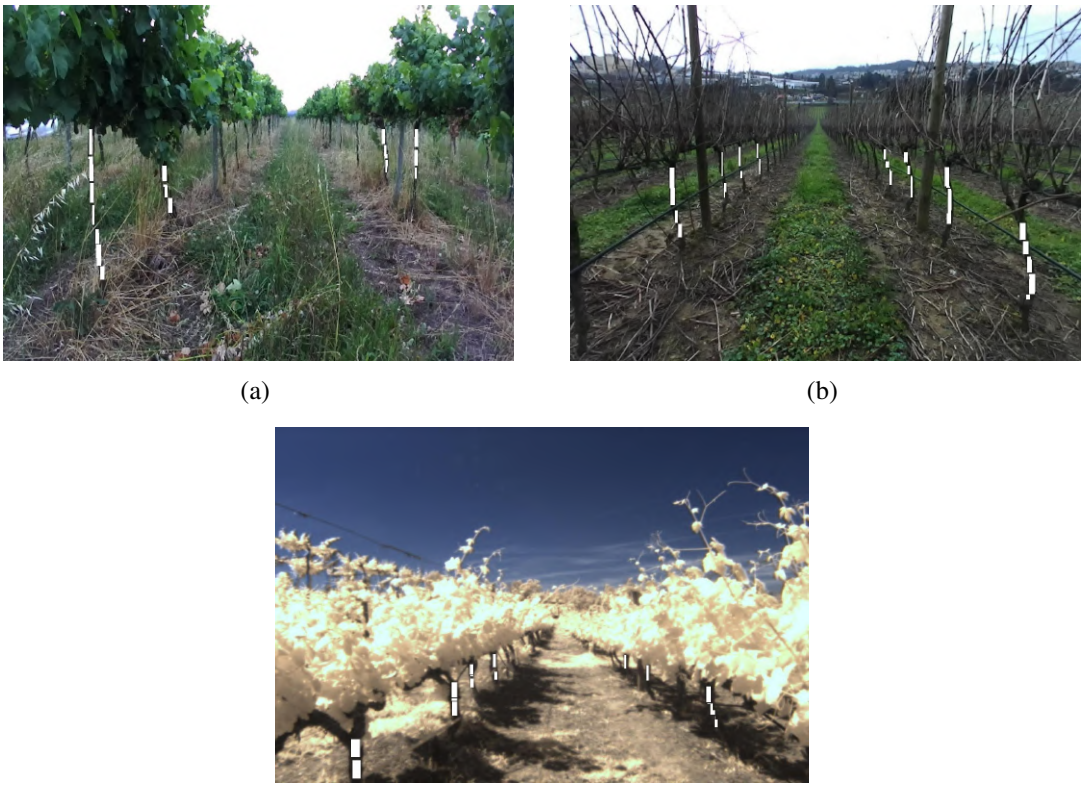
6.2 VineSet Trunk Segmentation using object detection models

Model	Resolution	Inference time (ms)	Fine-tuning		From scratch			
			50k		50k		100k	
			JSI(%)	DSI(%)	JSI(%)	DSI(%)	JSI(%)	DSI(%)
SSD MobileNet-V1	300 x 300	4.22	52.23	68.73	42.55	59.59	52.17	68.57
	512 x 512	10.90	-	-	43.75	60.87	54.77	70.78
SSD MobileNet-V2	300 x 300	4.65	50.56	67.23	49.15	65.90	50.76	67.34
	512 x 512	11.59	-	-	46.99	63.94	51.54	68.14
SSD Inception-V2	300 x 300	23.51	47.30	64.22	42.16	59.32	47.86	64.74
	512 x 512	32.48	-	-	48.39	65.22	49.27	66.01
SSD MobileNet-Vine	300 x 300	6.15	-	-	42.07	59.23	52.46	68.82
	512 x 512	13.81	-	-	43.31	60.45	51.72	68.18

Table 6.2: DICE similarity index (DSI)(%), Jaccard similarity index (JSI) (%) and average inference time per image (ms) using Coral Edge TPU, with fine-tuning and from scratch training.

To evaluate DL models for trunks segmentation on top of Google's Edge TPU the performance of the four models, i.e., SSD MobileNet-V1, SSD MobileNet-V2, SSD Inception-V2 and SSD MobileNet-Vine, with two different configurations (input size of 300x300 and 512x512) is compared. Also, some models were not trained with transfer learning due to the non-existence of

pre-trained models with those configurations. For this, was applied the same evaluation data used on section 6.1. However, with the annotations corresponding to the segmentation of the trunks. Therefore, the performance of the different detectors is evaluated using the Jaccard Similarity Index (JSI) and the Dice Similarity Index (DSI). As mentioned in equation 3.12 , DSI is very similar to the JSI. They are positively correlated, meaning that if one says that model X is better than model Y at segmenting a trunk, then the other says the same. So, these metrics show how good the overlap is between two binary images, i.e., ground truth and detection binary images. Also, the speed of the models, together with Edge TPU in the detection performance, was explored. Thus, the inference time is also evaluated. Table 6.2 summarizes the JSI, DSI and run time performance for all models configurations. Fig. 6.2 shows an example of trunk segmentation for three different vineyards, provided by SSD MobileNet-V1 with an input resolution of 512x512, that achieves the best performance.



(a) (b)

(c)

Figure 6.2: Segmentation results using SSD MobileNet-V1 (512x512 input size), where the white pixels are the True Positives.

In order to evaluate the segmentation performance with another state-of-the-art architecture, Faster R-CNN is trained with an input size of 1024x600 to use TL and because in Faster R-CNN paper [39] is mentioned that with this resolution, the architecture achieves better performance. However, this architecture is not compatible with Edge TPU since it is not quantized. Thus, it is evaluated on top of the Colab GPU. The same metrics used before are used here too. Table 6.3 summarizes the JSI, DSI, and run-time performance for Faster R-CNN.

Model	Resolution	Inference time (ms)	Fine-tuning		From scratch			
			200k		200k		400k	
			JSI(%)	DSI(%)	JSI(%)	DSI(%)	JSI(%)	DSI(%)
Faster-RCNN ResNet50	1024x600	102.70	60.70	75.54	48.39	65.22	59.97	74.98

Table 6.3: DICE similarity index (DSI)(%), Jaccard similarity index (JSI) (%) and average inference time per image (ms) using Colab GPU, with fine-tuning and from scratch training.

6.2.1 Discussion

The results on table 6.2 for a 300x300 resolution for both training with fine-tuning and from scratch shows that SSD MobileNet-V1 model obtained the best result for DSI with 68.73% similarity to the ground truth and JSI of 52.53% score. MobileNets achieve again together with Edge TPU the best inference times, which was expected to because these use depthwise separable convolutions and are lightweight models designed to run on mobile devices.

The input image dimension is an important parameter to take into consideration for the performance of the network. In the original paper of SSD, they recommend using a resolution of 512x512 in order to improve the detection of small objects. So, with a higher input dimension, the neural network has more spatial information about the image, as the produced feature maps have a bigger dimension when compared to the ones produced when the input dimension is smaller. With more spatial information, the network should be able to identify and extract the object features more clearly. However, if the information size is expanded, more operations need to be performed, leading to an increase in the inference time. In table 6.2 are present the values of the performance of the object detector models for trunk segmentation while varying the input image dimensions. The results confirm that with a higher input resolution, we have better results, increasing approximately 2% for all models compared to 300x300 resolution. The SSD MobileNet-V1 model achieves the best results again, having obtained 54.77% for JSI and 70.78% for DSI. The inference time increases as expected since a higher resolution input comes more operations, which implies a longer inference time. Although this inference time is higher, it is still possible to perform real-time detection due to the high-performance ML inferencing of Edge TPU. So, Edge TPU allows obtaining an excellent average inference time per frame of 10.90 ms for the SSD MobileNet-V1 model with a 512x512 input size, which results in a speed of 92 FPS. Concluding with this that it is possible to achieve good results segmenting the trunks together with real-time performance.

As previously stated, one of the significant issues regarding the SSD architecture was its inaccuracy in detecting and classifying small objects in an image due to spatial constraints. As mentioned in section 5.2.3, to address this problem, a layer with an increased feature map was added to the detection to improve this drawback. In table 6.2, we can see the results for SSD MobileNet-Vine, that is the modified model of SSD MobileNet-V1. The result of the modified model for an input resolution of 300x300 was slightly better than the original. However, comparing this resolution with the original model with an input size of 512x512, the result was smaller.

Which was expected since in the original model with a 512x512 input resolution increase all feature maps size, in the SSD MobileNet-Vine we only added a layer for detection with an increased feature map, keeping the size of the feature maps for input with a resolution of 300x300. The result of the modified model for a resolution of 512x512 was very similar to the original model with an input resolution of 300x300. However, comparing to the original model with an input resolution of 512x512, the result is smaller. So, besides an additional layer with a large feature map size used to make detections, all the feature maps size increase due to the higher input resolution, which could be the reason for lower performance. Also, the inference time increases for both resolutions not only because we have an additional layer with a large feature map size to make detections but also as a result of a larger input resolution. Nevertheless, it is still possible to carry out real-time detection with Edge TPU.

Looking at table 6.3 Faster R-CNN presents better results on smaller objects than SSD. The authors of SSD [13], said that this is probably because the RPN-based approaches use two shots. SSD only takes one single shot to detect multiple objects. However, RPN-based approaches such as R-CNN use two shots, one to create region proposals and other for detecting each proposal's object. Besides, since the previously used architectures are lightweight and dedicated for embedded and mobile devices, it is to be expected that this architecture will perform better on small objects. As expected, the results on table 6.3 show an improvement of 4%. Since this architecture is computationally more complex and is not compatible with Edge TPU, implies a high inference time. The average inference time per image result was 102.70 ms, which caps speed to 10 FPS, making it impossible for real-time performance due to the non-use of Coral Edge TPU.

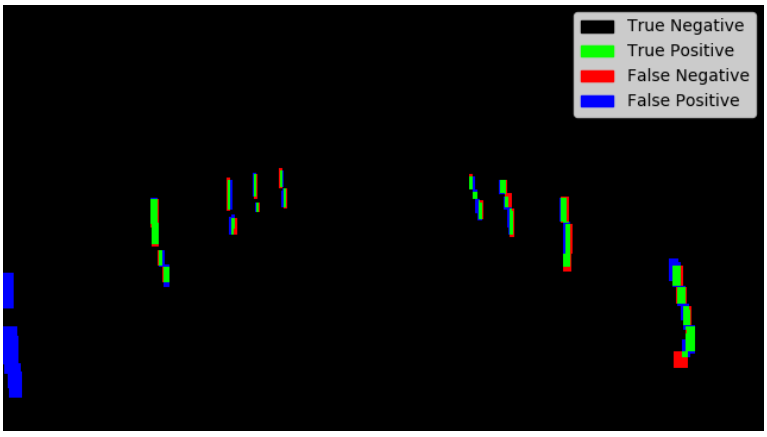


Figure 6.3: TP, TN, FP and FN representation for trunk segmentation.

Analysing the results in more detail, the presence of several pixels classified as False Positive can be seen in Fig. 6.3. However, many of these pixels should be detected as True Positive. This lack of matching is not only since trunk segmentation using bounding boxes is not much precise, leading to possible flaws in the complete trunk segmentation. But also due to a possible lack of trunk annotation for the ground truth. Thus, this proves that detections produce segmentations of trunks with higher quality than the manual annotations and the previous results could be improved if the ground truth annotations for trunk segmentation were improved too.

In short, a higher input size improves the performance of all models. Globally SSD MobileNet-V1 is the detector that achieves the best performance for trunk segmentation. Although the inference time is superior, the high performance of Edge TPU provides real-time performance. Furthermore, this approach similarly to section 6.1 can be used both on mapping and localisation task. However, this solution gives more details about the trunk, such as the shape. In this way, this approach gives a better input for SLAM algorithms in vineyards.

6.3 Semantic Segmentation

This approach was the last procedure of this dissertation and was made to verify the application of VineSet in semantic segmentation in possible future work. Initially, we have to reduce the train set to only two vineyards due to the images number limitation of Colab. So, to evaluate semantic segmentation models on top of Colab GPU the performance of two models, i.e., FC-DenseNet103 and DeepLabV3, are compared using an evaluation data composed by 300 images with approximately 850 vine trunks. For this, metrics of section 6.2 are used to here. The inference time is not evaluated since the application of the semantic segmentation models in an environment and configuration that supports real-time was left for future work. Table 6.4 shows the JSI and DSI for the two models. Fig. 6.4 shows an example of semantic segmentation result for FC-DenseNet103.

Model	**Crop size**	**From scratch**	
		138k	
		JSI(%)	**DSI(%)**
FC-DenseNet103	480x480	25.66	40.84
DeepLabV3	480x480	16.67	28.57

Table 6.4: DICE similarity index (DSI)(%), Jaccard similarity index (JSI) (%) and average inference time per image (ms) using Colab GPU with from scratch training.

(a) (b)

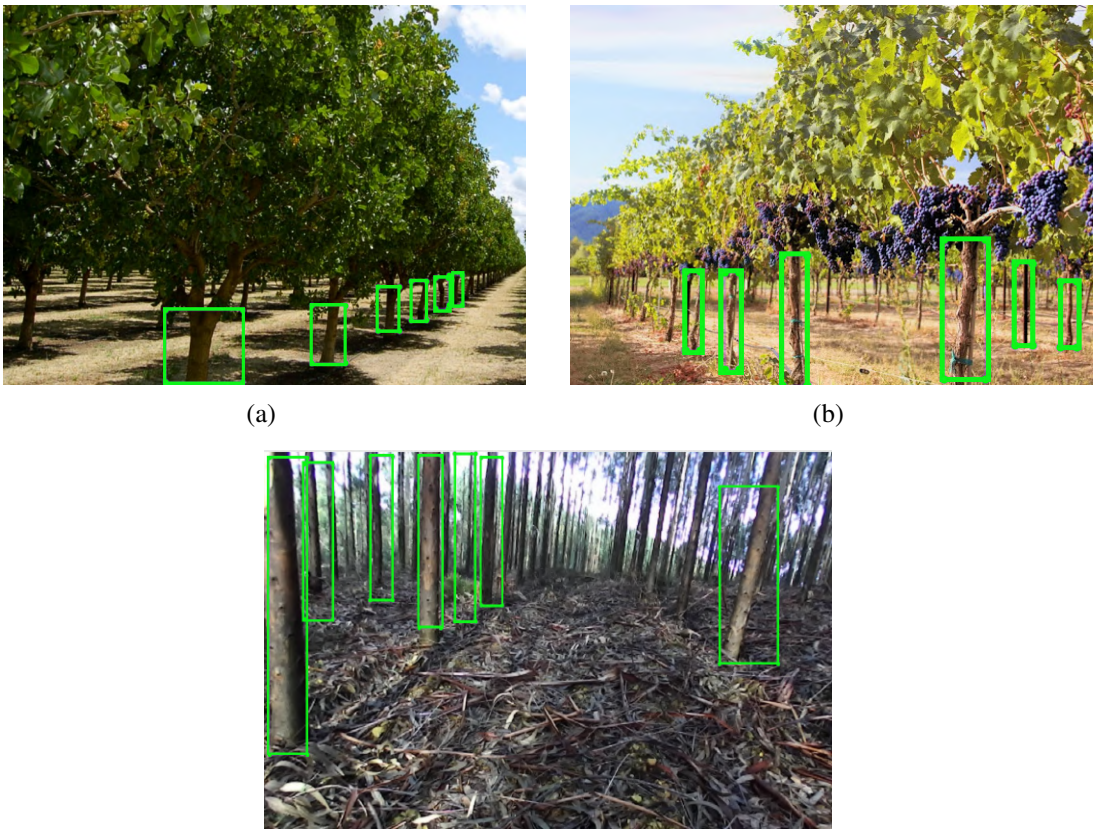
Figure 6.4: Semantic segmentation results using FC-DenseNet103, where the white pixels are the True Positives.

6.3.1 Discussion

Analysing table 6.4, DenseNet model shows better results than DeepLabV3. Also, table 6.4 show lower results than the obtained in section 6.2 using object detection models to segment the trunks. However, the results obtained here are not viable to make a comparison due to the limitations. One of the main reasons was that many fewer images were used for training due to the limitation of imagens in Colab. Another was because binary images resulting from the ground truth annotations of object detection models for trunk segmentation were used as ground truth segmentation mask. This ground truth masks need to be obtained with a pixel-wise image labelling tool, thus giving greater precision in the segmentation annotation, which improves the results.

In conclusion, although it is not possible to make a viable comparison, this step has been taken in order to open the door to future work using VineSet.

6.4 Assisted Labelling Procedure



(a) (b)

(c)

Figure 6.5: Automatic annotations result in different areas of agriculture such as (a) hazelnut orchard, (b) other vineyard and (c) forest.

In order to evaluate the performance of our Assisted Labelling procedure, four different datasets from distinct areas of agriculture are used to compare the automatic annotations and the time spent

with and without our tool for each dataset. Table 6.5 shows the number of images, number of trunks, automatic annotations, time spent with assisted labelling and without for each dataset.

Dataset	**Number of images**	**Number of trunks**	**Automatic annotations (%)**	**Time with assisted labelling (min)**	**Time without assisted labelling (min)**
VineSet	640	3648	89.04	33.71	304
Others vineyards	11	75	72.32	1.74	6.25
Ochards images	20	139	48.34	5.99	11.58
Forest images	264	1647	28.05	101.97	137.25

Table 6.5: Automatic annotation percentage and time of manually and assisted labelling with different agriculture areas.

Fig. 6.5 shows the result of our Assisted Labelling providing automatic annotations in different areas of agriculture such as others vineyards, orchards and forests.

6.4.1 Discussion

The results obtained in the table 6.5, regarding the creation of the assisted labelling procedure, show that VineSet, together with trained models, help in the detection and automatic labelling of trunks in vineyards. Also, this tool can be implemented in other Woody Crop areas of agriculture such as almond, apple, hazelnut and pistachio orchards or forests. However, we noticed that it was more efficient in the initial cultivation phase and also in trunks relatively similar to those of the vineyards. We can also see from the table 6.5 that the time spent using our labelling assistance tool is much less compared to manual labelling, reducing the time spent by approximately 72% in other vineyards, 52% in different orchards and 26% in forest images. So, our tool, in addition to creating automatic trunk labels without the user having to insert a model, can be used on any device, just having access to the Internet.

In conclusion, this approach presents a solution to reduce the time spent on the creation or improvement of a dataset not only in vineyards but also in other areas of agriculture. Thus, the benefit of using this tool is that it reduces the percentage of annotations taken manually, aiming to significantly reduce the time it takes to insert labels into relatively large datasets. It is worth noting that this procedure is iterative, in the way that the user can improve DL-based object detection models performance, by iteratively annotating objects that the model fails to recognize.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation is intended to respond to the lack of works carried out in the state-of-the-art to detect and segment the trunks in the vineyards using DL. For this, a low-power and high-performance DL-based to detect and segment vine trunks, capable of executing real-time operations in agriculture robotics were implemented. Moreover, this work aims to give a response to the five main questions initially raised in the problem description.

Firstly, in this work, we present the first dataset of trunk vineyards with the respective annotations accepted and published in the Robotic Operating System (ROS) Agriculture community (http://wiki.ros.org/agriculture), called VineSet. One of the innovations of VineSet is that it aims to provide a data benchmarking to create DL-based detection models according to realistic characteristics of the agricultural environment in vineyards. Taking into account **Question 1** of problem description, the creation of VineSet, with different data augmentation operations, shows that is robust to different conditions of outdoor environments and is capable of generalising to other agriculture areas. Besides, by analysing the validation loss during the training shows that the models avoid the overfitting. So, this approach solves the problem of **Question 1**.

The selection of the models to be trained was made taking into account the compatibility and size supported by the Edge TPU. So, all the models used together with Coral are lightweight and quantized. Even so, the results show that reliable detectors were obtained, suitable for execution on autonomous robotic platforms due to low energy consumption. Thus, this approach gives a response to the **Question 2**.

Furthermore, in order to achieve real-time performance object detection, Coral Edge TPU demonstrates not only that is low-cost hardware but also reduce the inference time significantly, showing that is possible to achieve real-time vine trunk detection. Therefore, due to the high-performance ML inferencing, Edge TPU proves that it is possible to combine real-time performance object detection with a low power processor. Thereby, this work allows the integration of edge-AI algorithms in SLAM, improving the input for the localisation and mapping of the robot,

through natural markers in the vineyards. Besides this, this approach gives a response to the **Question 3**.

In addition to this, we present an assisted labelling tool, that uses a model trained with VineSet, to reduce the time spent with manual annotation. The combination of automatic labelling tool with VineSet present good results in other environments such as orchards or forests. Thus, contributing to higher speed and efficiency in the annotation and implementation of future datasets for detecting trunks not only in the vineyards but also in the several areas of agriculture. This approach gives a solution to the **Question 4**.

Among this, DL-based object detection models, together with VineSet, also presents good results for the segmentation of trunks. The analyses of the results show that the detections produce segmentations of trunks with higher quality than the manual annotations and also shows that achieves real-time vine trunk segmentation, giving a good response to the **Question 5**.

Finally, semantic segmentation models were briefly addressed with the primary objective of verifying the use of VineSet for future work. During implementation, some limitations raised, therefore rendering any conclusion or comparison from results of semantic segmentation models non-viable. It is, however, safe to assume that the results show a possible path for future improvement. Besides, semantic segmentation can substitute DL-based object detection models for pixel-based classification if a proper training set is provided. Thus, overcoming these limitations would need to be performed on the training process to obtain a consistent result.

In conclusion, the main goals of this dissertation were achieved through the implementation process and results by giving response to the essential questions proposed by the problem definition present on this work and contributing for the implementation of DL in agriculture.

7.2 Future Work

In this dissertation, we took the first step to open a door in the use of semantic segmentation. The results show that semantic segmentation can substitute DL-based object detection models for pixel-based classification if a proper training set is provided. However, due to several limitations in training and since the VineSet annotations should be improved for use in semantic segmentation models, the results obtained did not allow us to make viable conclusions. As future work the main focus would be to improve ground truth annotations, i.e., ground truth segmentation masks, using tools that allow pixel-wise labelling, thus giving a better input for the training of the various semantic segmentation models. This task will improve robot mapping and location in several vineyards.

Another future task would be to create an online platform that would allow users to use our assisted labelling tool for trunk detection created in this work. At the same time, retain all the images inserted and respective annotations to a database, in order to extend VineSet even further.

Appendix A

Attachments

A.1 Submitted Papers

The following paper was submitted during this dissertation and is still in reviewing process.

VineSet: A Deep Learning-Oriented Woody Crops Trunk Image Collection and an Assisted Labelling procedure

Nuno Namora Monteiro[3], André Silva Aguiar[1,2], Filipe Neves dos Santos[2], and Armando Jorge Sousa[1,3]

[1] INESC TEC - Institute for Systems and Computer Engineering, Technology and Science, Porto, Portugal,
{andre.s.aguiar, fbsantos}@inesctec.pt
[2] UTAD - University of Trás-os-Montes e Alto Douro, Vila Real, Portugal,
[3] FEUP - Faculty of Engineering of University of Porto, Porto, Portugal,
{up201607764, asousa}@fe.up.pt

Abstract. Agricultural robots need image processing algorithms, which should be reliable under all weather conditions and be computationally efficient. Developing a system with a real-time performance for low-power processors is nowadays a research and development challenge because the lack of real data sets annotated and expedite tools to support this work. To support the deployment of deep-learning technology in agricultural robots, this paper presents a public VineSet (trunks dataset), the first public large collection of vine trunk images. The dataset was built from scratch, having a total of 9481 real image frames and providing the vine trunks annotations in each one of them. VineSet is composed of RGB and thermal images of 5 different Douro vineyards, with 952 originally collected by AgRob V16 robot, and others 8529 image frames resulting from a vast number of augmentation operations. To check the validity and usefulness of this VineSet dataset, in this paper is presented an experimental baseline study, using state-of-the-art Deep Learning models together with Google Tensor Processing Unit. To complement this dataset and simplify the task of this dataset augmentation by other groups, in this paper we propose an assisted labelling procedure - by using our trained models - to reduce the labelling time, in some cases ten times faster per frame. This paper presents preliminary results to support future research in this topic, for example with VineSet leads possible to train (by transfer learning procedure) existing deep neural networks with Average Precision (AP) higher than 80%. For example, an AP of 84.16% was achieved for SSD MobileNet-V1. Also, the models trained with VineSet present good results in other environments such as orchards or forests. Our automatic labelling tool proves this, reducing annotation time by more than 30% in various areas of agriculture and more than 70% on vineyards.

Keywords: Deep Learning · Agriculture · Image Processing.

1 Introduction

In the past few years, robotics as evolved exponentially, introducing itself as a significant tool in the execution of repetitive tasks. Robots appeared as a solution to many problems, among which stand out: on the one hand, the possibility to supplant the direct interaction of man in tasks where, in times, it was unreplaceable, contributing in this way to a marked decrease in work-related accidents; in other hand, robots gave way to the elimination of "dead times", due to their substantial autonomy and ability to act for several hours straight [1]. Therefore, it can be said that, in addition to solving various problems, they ensure greater efficiency and accuracy in the tasks they perform. Thus, robotics has a definite impact on the development of autonomous, independent, robust and efficient systems.

Since the productivity of agriculture, for most of history, is mainly the result of human action, as it still is today, it is necessary to invest in new processes that face this problem. In this sense, there has been an increase in the demand for robotic solutions to monitor and supervise agricultural crops [2]. Thus, new scientific fields arise, such as farming precision, also called digital agriculture, which boosts productivity and minimise environmental impact. As a basis, Machine Learning (ML) a subset of Artificial Intelligence (AI), is a mechanism that enables the machine to learn without being necessarily programmed, combined with new technologies and high-performance computing, which opens new horizons and makes work more efficient and effective [3]. Along with this, Deep Learning (DL) is a part of ML methods, that provides varied applications from natural language to image processing. DL architectures have been increasingly used in several research areas, including agriculture [4].

One of the tremendous current challenges of robotics for agriculture is to achieve image processing algorithms that are robust to all-weather illuminations conditions and at the same time, efficient and effective to be implemented in autonomous robots of small dimension and with limited energy capacity. Furthermore, several limitations may arise, such as the characteristic vineyard terrain irregularities or overfitting in the training of neural networks that may affect the performance. In parallel with this, the evolution of DL models became more and more complex, demanding an increasing computational complexity. Thus, not all processors can handle efficiently such models, and so, the combination of real-time performance with low power processing becomes demanding.

In this work, aiming to overcome the several limitations previous described and due to the non-existence of vine trunk datasets, we present a new public large collection of annotated vine trunks images called VineSet - collected using AgRob V16 robot [5,6] and fully described in section 3.

To simplify the process of dataset augmentation and reduce the labelling time, in section 4 is presented an assisted labelling procedure, build from the trained models.

In short, this work contributes to the development and improvement of current solutions in agriculture, since vision-based detectors provide complementary and richer information to reach a trustworthy natural feature detector [7]. For

example, will enable the integration of edge-AI in SLAM algorithms, like VineSLAM [5], which will serve for the localisation and mapping of the robot, through natural markers in the vineyards.

2 Related work

DL applications have been increasingly implemented in several areas of agriculture. However, because robotic platforms are typically electric and powered by low batteries, there may be some limitations due to the demand for hardware using the DL. In order to optimize and get the best performance, it is convenient to have as many information about the surrounding environment as possible. The most popular areas where DL is applied are for example plant disease detection [10,12,13], plant recognition [14], fruit counting [15,16], weed detection [17] and crop yield estimation [11]. From these, different DL architectures are choosen such as Convolutional Neural Network (CNN) [12,13,14,15][17], Faster R-CNN [10,11][16] and Single Shot Multibox Detector (SSD) [10,11]. Along with this, some of them used DL models like VGG, MobileNets, Darknet (YOLO), others created their own model. From this works, we can observe that most created their own dataset, others use dataset that are available to the public. Moreover, data augmentation is a usual practice among the majority to increase the dataset and enhance the results of the neural network. However, in most cases they have restrictions imposed by the complexity of DL model, being mostly used GPUs which impose large power requirements [18].

In order to provide better knowledge about complexity and efficiency of differents DL models, Bianco et al. [9] provided study over 40 different DL model for image recognition elements such as computational cost and accuracy on two distinct hardware platforms, NVIDIA Jetson TX1 and NVIDIA Titan X. This work shows that the recognition accuracy does not increase as the number of operations increases and not whole the DL models use their parameters with the same level of efficiency. We can also observe how efficiently each model uses its parameter and can notice that the models that use their parameters most efficiently are the SqueezeNets, ShuffleNet and the MobileNets. Besides that, they conclude that the model reaching the highest accuracy is the NASNet-A-Large, presenting, however, the highest computational complexity. Between all the models that have the lowest level of model complexity, SE-ResNeXt-50, has the highest accuracy and low level of model complexity.

Several tools for manual annotation in images are made available to the public, however, not all of these tools make automatic labels since they do not use AI. In [21], manual annotation software for image, audio and video is provided, running in a web browser and does not require installation or configuration. However, it does not allow the user to use a pre-trained model. On the other hand, tools such as DeepLabel (`https://github.com/jveitchmichaelis/deeplabel`) and Label Studio (`https://github.com/heartexlabs/label-studio`) allow us to label from a pre-trained model automatically, however, if the user has no model, the annotation is done manually.

This work aims to fill a gap of the state-of-the-art, proposing the first public vine trunk image collection, named VineSet. Besides being a vast image collection, VineSet is also diverse, providing images and annotations from five different Douro vineyards, including RGB and thermal formats. This paper also presents an assisted labelling procedure to reduce the time spent on traditional labelling methods. To overcome one of the main obstacles of combining real-time performance with a low power processor, the TensorFlow framework is combined with the most recent TPU of Google, the Coral Edge TPU. This device provides high-performance DL inference with low-power costs. Furthermore, we provide an evaluation of the performance of different DL models on the detection of trunks in vineyards using the VineSet.

3 VineSet

The detection of trunks in vineyards using Deep Learning algorithms is an area yet to be explored. VineSet, an open-source dataset, was published in the Robotic Operating System (ROS) Agriculture community (`http://wiki.ros.org/agriculture`). This dataset was created using Agrob V16 (Fig. 1), a robotic platform for research and development of robotics technologies for Douro vineyards, that captured images containing vine trunks in different contexts and in different formats.

Fig. 1: AgRob V16 robotic platform.

In the creation of VineSet, several steps were outlined, due to the non-existence of public datasets related to the detection of trunks in the vineyards. The first step went through by collecting images of 5 different vineyards, and

then annotations were made in all images and, finally, data augmentation was addressed to increase the dataset size.

3.1 Data collection

The VineSet, contains images with several Douro Vineyards. The data gathering was done under different stages of crop process and different times of the day, thus changing the lighting conditions. One of the most exclusive characteristics of VineSet is the fact that it contains thermal images, which are not included in many other datasets. This was possible since our robotic platform is equipped with both an RGB and a thermal camera, which allows collecting a larger sample, thus increasing, even more, the diversity of images. The advantage of using thermal images is the fact that they reduce the variability of images under different lighting conditions, thus allowing the detection of trunks to be more accurate.

During the data collection procedure, Agrob V16 collected several video streams in each vineyard. Since the training input is a set of images, 952 samples were extracted from them.

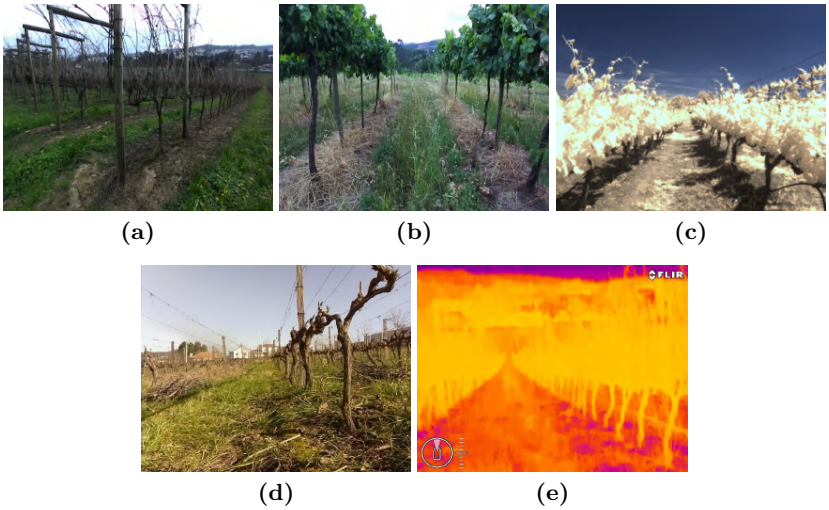


Fig. 2: Set of different vineyards used in the VineSet data collection.

3.2 Data annotation

From the data collection, trunks were manually annotated for all images, using the PASCAL VOC format, each of which represented a bounding box associated with the respective class. All of these annotations are included in VineSet. In

some cases, the vegetation in the vineyards causes obstructions in the identification of the trunks. Therefore, trunks that are not clear to distinguish with the surrounding environment were not included in this annotation process. This process aims to label the class and location coordinates of each of the bounding boxes that contain the region of interest that in this case are the trunks. In this way, we are able to have a ground-truth that allows us to evaluate the prediction results during the various tests.

3.3 Data augmentation

Even though DL outperforms most traditional machine learning methods in terms of precision and real-time application [4], one of the biggest challenges is to overcome overfitting. This is one of the frequent problems in ML that consists of modelling the data too well, learning only the expected output for each input instead of learning the general distribution of the input data. In addition to this, we have conditions such as variation of sunlight illumination during the day or the terrain of outdoor environments that may affect performance. In this way, to avoid overfitting and generalise the network, data augmentation is a usual method to enhance the variability of data for training by enlarging the dataset using label-preserving transformations.

(a) Translation (b) Multiply (c) Hue and Saturation

(d) Horizontal Flip (e) Rotation -15 degrees (f) Rotation 15 degrees

Fig. 3: Set of several augmentation operations used to expand VineSet.

In order to increase the diversity of the VineSet, the collected images were pre-processed with typical augmentation techniques used in the computer vision, such as horizontal flipping, image re-scaling, rotation, translation, multi-

ply, Gaussian noise, hue and saturation. Some of the augmentation operations performed are shown in Fig. 3.

4 Deep Learning-based Assisted Labelling

Training a DL model involves several steps, one of the most important of which is data annotation. Generally, this step is a long process, and the time spent depends on several factors, such as the total number of images that the dataset has, the number of classes and the ease of manually identifying the bounding box corresponding to each class. Thus, this paper proposes to create an assisted labelling procedure that uses AI to help the annotation process in the detection of trunks in the vineyards. The layout of the created application is represented in Fig. 4.

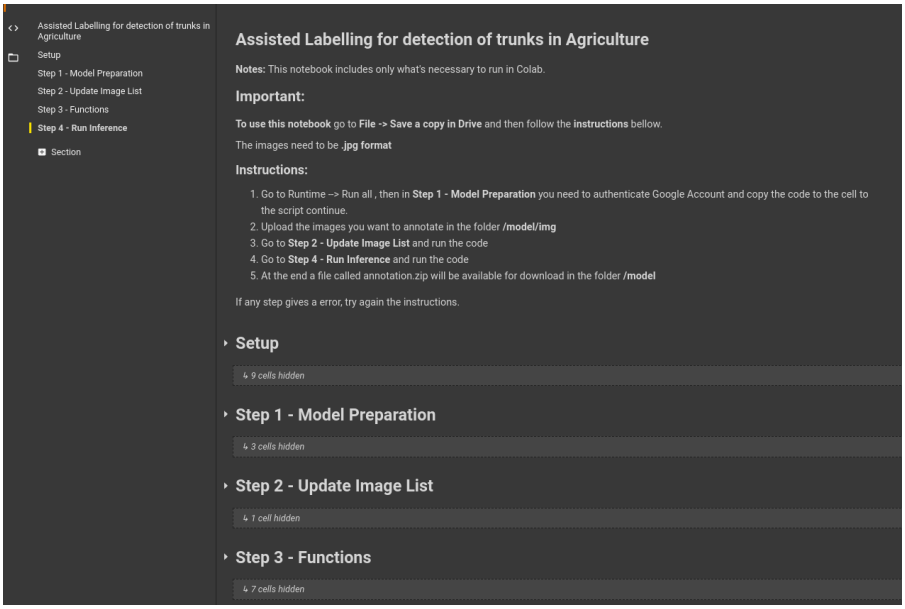


Fig. 4: Assisted labelling interface.

In this way, a python notebook is made available in the CRIIS repository (`https://gitlab.inesctec.pt/agrob/agrob_vineset`), which is open to the general community. The procedure of this new solution consists of using an online platform, Google Colaboratory (`https://colab.research.google.com`), so that the user can save the resources of his machine. This tool provides a DL model trained for detecting vine trunks, but also capable of detecting trunks in other contexts such as orchards or forests. So, an essential factor for automating

this process is the use of the DL model. Taking into account the results obtained on section 5.1, the SSD MobileNet-V1 trained with VineSet was the model chosen for the detection of trunks in the images introduced in this tool. The assisted labelling procedure uses this model to pre-process the user dataset, automatically annotating the detected trunks, saving the annotations in the Pascal VOC format. The user can then load the automatic annotations and complete them manually. One of the benefits of using this tool is that it reduces the percentage of annotations taken manually, aiming to significantly reduce the time it takes to insert labels into relatively large datasets. It is worth noting that this procedure is iterative, in the way that the user can improve DL-based object detection models performance, by iteratively annotating objects that the model fails to recognize.

5 Results

To compare the performance of state-of-the-art DL models on VineSet dataset, it is necessary to have an evaluation method in order to draw conclusions. In this paper, the performance of our system is evaluated using the average precision (AP) based on intersection-over-union (IoU) introduced in the PASCAL VOC Challenge [19]. Thus, the concept of IoU is fundamental in the use of these metrics, that can be expressed as the overlap ratio between the ground-truth bounding box (B_{gt}) and the predicted bounding box (B_p) result of neural network as follows:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{1}$$

As in the Pascal VOC Challenge, the AP is a set of eleven equally spaced recall levels $[0, 0.1, ..., 1]$, and the mAP is the AP computed over all classes, in this case, the trunk class.

$$AP = \frac{1}{11} \sum_{r \in [0, 0.1, ..., 1]} p_{interp}(r) \tag{2}$$

$$p_{interp}(r) = \underset{\widetilde{r}, \widetilde{r} \geq r}{max}\, p(\widetilde{r}) \tag{3}$$

Another metric used was the F1-Score, which aims to measure the harmonic mean between precision and recall.

$$F_1 = 2 \times \frac{precsion \times recall}{precision + recall} \tag{4}$$

In the other hand, to evaluate the performance of our assisted labelling procedure, a series of experiments were conducted with some different dataset size of different agriculture areas that contains unseen data, in order to verify the efficiency and compare the time spent between the assisted labelling and manual labelling.

5.1 VineSet experimental deployment

To verify the performance of different models with VineSet, our dataset, has been divided into 90% training set and 10% testing set. Also, experiments were carried out training the models from scratch (random weight initialization) and using transfer learning (initialized with pre-trained weights) to analyze which of these would reach a faster satisfactory result. The detection models were trained on an online platform, Colab, with an NVIDIA Tesla P100-PCIE. Taking into account memory and time constraints, hyperparameters have been adjusted. Thus, the batch size was set to 18, and the number of training steps was 50,000. To verify the performance of models trained from scratch, the number of training steps were increased to 100,000. Parameters such as momentum, learning rate and others, have not been changed, being the same as the original model.

Fig. 5: Detection results using SSD MobileNet-V1

Then, we compared the estimated results with the ground-truth using an IoU threshold set to 0.5. For this, metrics like AP and F1-Scores are used. Also, the speed of the dataset in the detection performance was explored, and one of the objectives of the creation of VineSet is to achieve an autonomous robotic platform that is robust, efficient, and that has real-time performance. Thus, the indicator chosen to evaluate the performance of the speed of each DL model was the inference time. In order to improve the real-time performance, the Coral Edge TPU was used. Compatibility and capacity of the Google accelerator were taking into account while choosing the models. The detection performance evaluation are shown in Table 1. Figure 5 shows three examples of the detections.

Model	α	Resolution	Inference time (ms)	Fine-tuning		From scratch			
				50k		50k		100k	
				AP(%)	F1	AP(%)	F1	AP(%)	F1
SSD MobileNet-V1	1	300 x 300	4.55	84.16	0.841	68.44	0.685	85.93	0.834
SSD MobileNet-V2	1	300 x 300	5.04	83.01	0.808	60.44	0.639	83.70	0.812
SSD Inception-V2	1	300 x 300	23.82	75.78	0.848	58.05	0.658	76.77	0.849

Table 1: AP (%), F1 Scores and average inference time per image (ms) using Coral Edge TPU, with fine-tuning and from scratch training.

5.2 Assisted labelling procedure

To assess the performance of our assisted labelling procedure tests were carried out in which several factors were analyzed in comparison with manual annotation. In order to draw conclusions, the average time to manually label a trunk was measured over several experiments, resulting in 5 seconds per trunk. Thus, once this value is established, it will be essential to calculate the total time spent on several images. The time spent on assisted annotation was calculated from the percentage of annotations made automatically and the percentage of annotations made manually. In this way, the total time spent by the tool is calculated through the time spent by the automatic annotation plus the offset created by the missing annotations.

Dataset	Number of images	Number of trunks	Automatic annotations (%)	Time with assisted labelling (min)	Time without assisted labelling (min)
VineSet	640	3648	89.04	33.71	304
Others vineyards	11	75	72.32	1.74	6.25
Ochards images	20	139	48.34	5.99	11.58
Forest images	264	1647	28.05	101.97	137.25

Table 2: Automatic annotation percentage and time of manually and assisted labelling with different agriculture areas.

In order to assess the scope of our tool, which was trained using VineSet, it was implemented in other areas of agriculture in addition to the vineyards. For this, we organize a range of images from different orchards and forests, where we evaluate the performance of our assisted labelling compared to the manual one. The results obtained in table 2 and figures show that our tool together with VineSet is able to cover some other areas of agriculture. Figure 6 shows two examples of automatic labelling.

(a) Hazelnut Orchard (b) Other Vineyard

Fig. 6: Automatic annotations in different areas of agriculture

5.3 Discussion

The results of VineSet deployment allowed us to make conclusions about the different models trained with our dataset. The comparison on Table 1 for training with fine-tuning shows that the SSD MobileNet-V1 model obtained the best AP result of 84.16%. Although the SSD MobileNet-V2 has a slightly lower AP result than the SSD MobileNet-V1, they are very similar, which was expected, since they have a very similar architecture. Also, they obtained the best time of inference, which is due to the fact that Mobilenet uses depthwise separable convolution, while Inception uses standard convolution. This results in fewer parameters on MobileNet compared to Inception V2. However, sometimes this results in a slight decrease in performance as well, which in this case was not verified. F1 scores show us that the best result is obtained by SSD Inception-V2. This is due to the fact that it has higher precision and a slightly lower recall compared to other models. Considering that this work uses lightweight models oriented to embedded devices, the AP results revealed to be entirely satisfactory on the object detection procedure. In this way, were obtained reliable detectors, suitable for execution on autonomous robotic platforms due to low energy consumption. Regarding training from scratch, the number of steps had to be doubled to achieve similar or better results than fine-tuning. With only 50,000 steps, the performance obtained is significantly lower, as shown in Table 1. This leads to the conclusion that, when training from scratch, the model needs to learn low and high-level semantics, so more iterations are needed to converge correctly. Thus, more effort is required in order to achieve satisfactory results [20].

The results obtained in the table 2, regarding the creation of the assisted labelling procedure, show that VineSet, together with trained models, help in the detection and automatic labelling of trunks in vineyards. Also, this tool can be implemented in other Woody Crop areas of agriculture such as almond, apple, hazelnut and pistachio orchards or forests. However, we noticed that it was more efficient in the initial cultivation phase and also in trunks relatively

similar to those of the vineyards. We can also see from the Table 2 that the time spent using our labelling assistance tool is much less compared to manual labelling, reducing the time spent by approximately 72% in other vineyards, 52% in different orchards and 26% in forest images. Compared to all other state-of-the-art tools, ours, in addition to creating automatic trunk labels without the user having to insert a model, can be used on any device, just having access to the Internet.

6 Conclusion

In this work, we present the first dataset of trunk vineyards with the respective annotations available to the scientific community, called VineSet. One of the innovations of VineSet is that it aims at providing the data benchmark to constructing Deep Learning-based detection models according to realistic characteristics of the agricultural environment in vineyards. To verify the performance of different models with VineSet, the Coral Edge TPU was used, which in addition to being low-cost hardware proved to reduce the inference time significantly. We concluded that our dataset leads low-cost models to achieve high-performance results. In addition to this, we present an assisted labelling tool, that uses a model trained with VineSet, to reduce the time spent with manual annotation. The combination of automatic labelling tool with VineSet present good results in other environments such as orchards or forests. Thus, contributing to higher speed and efficiency in the annotation and implementation of future datasets for detecting trunks not only in the vineyards but also in the several areas of agriculture.

For future work, we aim to expand further VineSet, including segmentation of trunks to improve the robot's performance and location in several vineyards. Also, we plan to retain the images inserted in the automatic annotation platform in a database for later insertion into VineSet.

References

1. Lyons, Siobhan. Death and the Machine, Intersections of Mortality and Robotics. Palgrave Pivot, 2018. doi:https://doi.org/10.1007/978-981-13-0335-7 .
2. Duckett, Tom and Pearson, Simon and Blackmore, Simon and Grieve, Bruce. (2018). Agricultural Robotics: The Future of Robotic Agriculture.
3. Royal Society. Machine learning : the power and promise of computers that learn by example, volume 66. 2017.
4. Santos, Luís and Neves Dos Santos, Filipe and Moura Oliveira, Paulo and Shinde, Pranjali. (2020). Deep Learning Applications in Agriculture: A Short Review.
5. Neves, Filipe and Neves Dos Santos, Filipe and Sobreira, Heber and Campos, Daniel and Morais, R. and Moreira, A. and Contente, Olga. (2015). Towards a Reliable Monitoring Robot for Mountain Vineyards. Proceedings - 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2015. 10.1109/ICARSC.2015.21.

6. Santos, L., Santos, F., Mendes, J., Costa, P., Lima, J., Reis, R., Shinde, P. (2020). Path Planning Aware of Robot's Center of Mass for Steep Slope Vineyards. Robotica, 38(4), 684-698. doi:10.1017/S0263574719000961
7. J. Mendes, F. N. d. Santos, N. Ferraz, P. Couto and R. Morais, "Vine Trunk Detector for a Reliable Robot Localization System," 2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC), Braganca, 2016, pp. 1-6.
8. Canziani, Alfredo and Paszke, Adam and Culurciello, Eugenio. (2016). An Analysis of Deep Neural Network Models for Practical Applications.
9. Bianco, Simone and Cadène, Rémi and Celona, Luigi and Napoletano, Paolo. (2018). Benchmark Analysis of Representative Deep Neural Network Architectures. IEEE Access. 6. 64270-64277. 10.1109/ACCESS.2018.2877890.
10. Fuentes, Alvaro and Yoon, Sook and Kim, Sang and Park, Dong. (2017). A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition. Sensors. 17. 2022. 10.3390/s17092022.
11. Heinrich, Kai and Roth, Andreas and Breithaupt, Lukas and Möller, Björn and Maresch, Johannes. (2019). Yield Prognosis for the Agrarian Management of Vineyards using Deep Learning for Object Counting.
12. Rançon, Florian and Bombrun, Lionel and Keresztes, Barna and Germain, Christian. (2018). Comparison of SIFT Encoded and Deep Learning Features for the Classification and Detection of Esca Disease in Bordeaux Vineyards. Remote Sensing. 11. 1. 10.3390/rs11010001.
13. Mohanty, Sharada and Hughes, David and Salathe, Marcel. (2016). Using Deep Learning for Image-Based Plant Disease Detection. Frontiers in Plant Science. 7. 10.3389/fpls.2016.01419.
14. H. Yalcin and S. Razavi, "Plant classification using convolutional neural networks," 2016 Fifth International Conference on Agro-Geoinformatics (Agro-Geoinformatics), Tianjin, 2016, pp. 1-5.
15. Koirala, Anand and Walsh, Kerry and Wang, Zhenglin and McCarthy, C.. (2019). Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'MangoYOLO'. Precision Agriculture. 10.1007/s11119-019-09642-0.
16. S. Bargoti and J. Underwood, "Deep fruit detection in orchards," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 3626-3633.
17. Ferreira, Alessandro and Freitas, Daniel and Silva, Gercina and Pistori, Hemerson and Folhes, Marcelo. (2017). Weed detection in soybean crops using ConvNets. Computers and Electronics in Agriculture. 143. 314-324. 10.1016/j.compag.2017.10.027.
18. C. Lammie, A. Olsen, T. Carrick and M. Rahimi Azghadi, "Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge," in IEEE Access, vol. 7, pp. 51171-51184, 2019.
19. Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. International Journal of Computer Vision, 111(1), 98-136, 2015
20. Kaiming He, Ross Girshick, Piotr Dollar; The IEEE International Conference on Computer Vision (ICCV), 2019, pp. 4918-4927
21. Abhishek Dutta and Andrew Zisserman. 2019. The VIA Annotation Software for Images, Audio and Video. In Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3343031.3350535

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. doi:10.1109/TKDE.2009.191.

[3] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, 2014. doi:10.1109/CVPR.2014.222.

[4] Konstantinos G. Liakos, Patrizia Busato, Dimitrios Moshou, Simon Pearson, and Dionysis Bochtis. Machine learning in agriculture: A review. *Sensors (Switzerland)*, 18(8):1–29, 2018. doi:10.3390/s18082674.

[5] Rodrigo M.S. De Oliveira, Ramon C.F. Araújo, Fabrício J.B. Barros, Adriano Paranhos Segundo, Ronaldo F. Zampolo, Wellington Fonseca, Victor Dmitriev, and Fernando S. Brasil. A system based on artificial neural networks for automatic classification of hydro-generator stator windings partial discharges. *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, 16(3):628–645, 2017. doi:10.1590/2179-10742017v16i3854.

[6] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, 2014. arXiv:1405.0312, doi:10.1007/978-3-319-10602-1_48.

[7] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016. doi:10.1016/j.neucom.2015.09.116.

[8] D. Gupta. (2019). A Beginner's guide to Deep Learning based Semantic Segmentation using Keras. Accessed: May 30, 2020. [Online]. Available: https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html.

[9] Zhong Qiu Zhao, Peng Zheng, Shou Tao Xu, and Xindong Wu. Object Detection with Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019. arXiv:1807.05511, doi:10.1109/TNNLS.2018.2876865.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. arXiv:1311.2524, doi:10.1109/CVPR.2014.81.

[11] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015. arXiv:1504.08083, doi:10.1109/ICCV.2015.169.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788, 2016. doi:10.1109/CVPR.2016.91.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016. arXiv:1512.02325, doi:10.1007/978-3-319-46448-0_2.

[14] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018. arXiv:1810.00736, doi:10.1109/ACCESS.2018.2877890.

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. URL: http://arxiv.org/abs/1704.04861, arXiv:1704.04861.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:2818–2826, 2016. arXiv:1512.00567, doi:10.1109/CVPR.2016.308.

[17] Google. (2019). Tensorflow Models on the Edge TPU. Accessed: May 10, 2020. [Online]. Available: https://coral.ai/docs/edgetpu/models-intro/.

[18] Siobhan Lyons. *Death and the Machine, Intersections of Mortality and Robotics*. Palgrave Pivot, 2018. doi:https://doi.org/10.1007/978-981-13-0335-7.

[19] Tom Duckett, Simon Pearson, Simon Blackmore, Bruce Grieve, Wen-Hua Chen, Grzegorz Cielniak, Jason Cleaversmith, Jian Dai, Steve Davis, Charles Fox, Pål From, Ioannis Georgilas, Richie Gill, Iain Gould, Marc Hanheide, Alan Hunter, Fumiya Iida, Lyudmila Mihalyova, Samia Nefti-Meziani, Gerhard Neumann, Paolo Paoletti, Tony Pridmore, Dave Ross, Melvyn Smith, Martin Stoelen, Mark Swainson, Sam Wane, Peter Wilson, Isobel Wright, and Guang-Zhong Yang. Agricultural Robotics: The Future of Robotic Agriculture. 2018. URL: http://arxiv.org/abs/1806.06762, arXiv:1806.06762.

[20] Royal Society. *Machine learning : the power and promise of computers that learn by example*, volume 66. 2017. URL: https://royalsociety.org/~/media/policy/projects/machine-learning/publications/machine-learning-report.pdf.

[21] Filipe Neves dos Santos, Heber Sobreira, Daniel Campos, Raul Morais, António Paulo Moreira, and Olga Contente. Towards a Reliable Robot for Steep Slope Vineyards Monitoring. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 83(3-4):429–444, 2016. doi:10.1007/s10846-016-0340-5.

[22] Luís Santos, Filipe Santos, Jorge Mendes, Pedro Costa, José Lima, Ricardo Reis, and Pranjali Shinde. Path Planning Aware of Robot's Center of Mass for Steep Slope Vineyards. *Robotica*, 2019. doi:10.1017/S0263574719000961.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. URL: http://dx.doi.org/10.1007/s11263-015-0816-y, arXiv:1409.0575, doi:10.1007/s11263-015-0816-y.

[24] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. doi:10.1007/s11263-009-0275-4.

[25] Andreas Kamilaris and Francesc X. Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, 147(February):70–90, 2018. doi:10.1016/j.compag.2018.02.016.

[26] Sergio Lima Netto Rafael Padilla and Eduardo A. B. da Silva. Survey on performance metrics for object-detection algorithms. 2020.

[27] Yutaka Sasaki. The truth of the F-measure. *Teach Tutor mater*, pages 1–5, 2007. URL: http://www.cs.odu.edu/{~}mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07.pdf.

[28] A. Koirala, K. B. Walsh, Z. Wang, and C. McCarthy. Deep learning for real-time fruit detection and orchard fruit load estimation: benchmarking of 'MangoYOLO'. *Precision Agriculture*, 20(6):1107–1135, 2019. URL: https://doi.org/10.1007/s11119-019-09642-0, doi:10.1007/s11119-019-09642-0.

[29] Aaron Carass, Snehashis Roy, Adrian Gherman, Jacob C. Reinhold, Andrew Jesson, Tal Arbel, Oskar Maier, Heinz Handels, Mohsen Ghafoorian, Bram Platel, Ariel Birenbaum, Hayit Greenspan, Dzung L. Pham, Ciprian M. Crainiceanu, Peter A. Calabresi, Jerry L. Prince, William R.Gray Roncal, Russell T. Shinohara, and Ipek Oguz. Evaluating White Matter Lesion Segmentations with Refined Sørensen-Dice Analysis. *Scientific Reports*, 10(1):1–19, 2020. doi:10.1038/s41598-020-64803-w.

[30] Syed Furqan Qadri, Danni Ai, Guoyu Hu, Mubashir Ahmad, Yong Huang, Yongtian Wang, and Jian Yang. Automatic deep feature learning via patch-based deep belief network for vertebrae segmentation in CT Images. *Applied Sciences (Switzerland)*, 9(1), 2018. doi:10.3390/app9010069.

[31] Achim Walter, Robert Finger, Robert Huber, and Nina Buchmann. Smart farming is key to developing sustainable agriculture. *Proceedings of the National Academy of Sciences of the United States of America*, 114(24):6148–6150, 2017. doi:10.1073/pnas.1707462114.

[32] Luís Santos, Filipe Neves dos Santos, Paulo Moura Oliveira, and Pranjali Shinde. Deep Learning Applications in Agriculture: A Short Review. *Advances in Intelligent Systems and Computing*, 1092 AISC(January):C1, 2020. doi:10.1007/978-3-030-35990-4.

[33] A. Kamilaris and F. X. Prenafeta-Boldú. A review of the use of convolutional neural networks in agriculture. *Journal of Agricultural Science*, 156(3):312–322, 2018. doi:10.1017/S0021859618000436.

[34] Michael A. Nielson. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/.

[35] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8695 LNCS(PART 7):297–312, 2014. arXiv:1407.1808, doi:10.1007/978-3-319-10584-0_20.

[36] G. Geetharamani and J. Arun Pandian. Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers and Electrical Engineering*, 78:536, 2019. URL: https://doi.org/10.1016/j.compeleceng.2019.08.010, doi:10.1016/j.compeleceng.2019.08.010.

[37] Alex Krizhevsky, Ilya Sutskever, and Geo Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Handbook of Approximation Algorithms and Metaheuristics*, pages 1–1432, 2007. doi:10.1201/9781420010749.

[38] J. R.R. Uijlings, K. E.A. Van De Sande, T. Gevers, and A. W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. doi:10.1007/s11263-013-0620-5.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. arXiv:1506.01497, doi:10.1109/TPAMI.2016.2577031.

[40] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019. doi:10.1109/ACCESS.2019.2939201.

[41] Joseph Redmon and Ali Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6517–6525, 2017. arXiv:1612.08242, doi:10.1109/CVPR.2017.690.

[42] Joseph Redmon and Ali Farhadi. YOLO v.3. *Tech report*, pages 1–6, 2018. URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[43] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, High-Quality Object Detection. 2014. URL: http://arxiv.org/abs/1412.1441, arXiv:1412.1441.

[44] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional Single Shot Detector. 2017. URL: http://arxiv.org/abs/1701.06659, arXiv:1701.06659.

[45] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. pages 1–7, 2016. URL: http://arxiv.org/abs/1605.07678, arXiv:1605.07678.

[46] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning Transferable Architectures for Scalable Image Recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. arXiv:1707.07012, doi:10.1109/CVPR.2018.00907.

[47] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation Networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. arXiv:1709.01507, doi:10.1109/CVPR.2018.00745.

[48] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. pages 1–13, 2016. URL: http://arxiv.org/abs/1602.07360, arXiv:1602.07360.

[49] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. arXiv:1707.01083, doi:10.1109/CVPR.2018.00716.

[50] Yu Emma Wang, Gu-Yeon Wei, and David Brooks. Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. 2019. URL: http://arxiv.org/abs/1907.10701, arXiv:1907.10701.

[51] Suchet Bargoti and James Underwood. Deep fruit detection in orchards. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3626–3633, 2017. arXiv:1610.03677, doi:10.1109/ICRA.2017.7989417.

[52] Maryam Rahnemoonfar and Clay Sheppard. Deep count: Fruit counting based on deep simulated learning. *Sensors (Switzerland)*, 17(4):1–12, 2017. doi:10.3390/s17040905.

[53] Steven W. Chen, Shreyas S. Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J. Taylor, and Vijay Kumar. Counting Apples and Oranges with Deep Learning: A Data-Driven Approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017. doi:10.1109/LRA.2017.2651944.

[54] Alvaro Fuentes, Sook Yoon, Sang Cheol Kim, and Dong Sun Park. A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition. *Sensors (Switzerland)*, 17(9), 2017. doi:10.3390/s17092022.

[55] Bin Liu, Yun Zhang, Dong Jian He, and Yuxiang Li. Identification of apple leaf diseases based on deep convolutional neural networks. *Symmetry*, 10(1), 2018. doi:10.3390/sym10010011.

[56] Mostafa Mehdipour Ghazi, Berrin Yanikoglu, and Erchan Aptoula. Plant identification using deep neural networks via optimization of transfer learning parameters. *Neurocomputing*, 235(January):228–235, 2017. URL: http://dx.doi.org/10.1016/j.neucom.2017.01.018, doi:10.1016/j.neucom.2017.01.018.

[57] Abraham George Smith, Jens Petersen, Raghavendra Selvan, and Camilla Ruø Rasmussen. Segmentation of roots in soil with U-Net. *Plant Methods*, 16(1):1–13, 2020. arXiv:1902.11050, doi:10.1186/s13007-020-0563-0.

[58] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. arXiv:1712.05877, doi:10.1109/CVPR.2018.00286.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:770–778, 2016. arXiv:1512.03385, doi:10.1109/CVPR.2016.90.

[60] Simon Jegou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017-July:1175–1183, 2017. arXiv:1611.09326, doi:10.1109/CVPRW.2017.156.

[61] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. 2017. URL: http://arxiv.org/abs/1706.05587, arXiv:1706.05587.

[62] Kaiming He, Ross Girshick, and Piotr Dollar. Rethinking imageNet pre-training. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob(ii):4917–4926, 2019. arXiv:1811.08883, doi:10.1109/ICCV.2019.00502.