Pontifícia Universidade Católica
do Rio de Janeiro

Gabriel Lins Tenório

Applications of Deep Learning for Crop Monitoring: Classification of Crop Type, Health and Maturity

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica.

Advisor: Prof. Wouter Caarls

Rio de Janeiro
April 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Gabriel Lins Tenório

Applications of Deep Learning for Crop Monitoring: Classification of Crop Type, Health and Maturity

Dissertation presented to the Programa de Pós–graduação em Engenharia Elétrica da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia Elétrica. Approved by the undersigned Examination Committee.

Prof. Wouter Caarls
Advisor
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Antonio Candea Leite
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Raul Queiroz Feitosa
Departamento de Engenharia Elétrica – PUC-Rio

Prof. Leonardo A. F. Mendoza
Universidade do Estado do Rio de Janeiro – UERJ

Rio de Janeiro, April 26th, 2019

Gabriel Lins Tenório

Majored in Control and Automation Engineering at the Pontifical University Catholic of Rio de Janeiro

Bibliographic data

Tenorio, Gabriel Lins

Applications of Deep Learning for Crop Monitoring: Classification of Crop Type, Health and Maturity / Gabriel Lins Tenório; advisor: Wouter Caarls. – Rio de janeiro: PUC-Rio , Departamento de Engenharia Elétrica, 2019.

v., 112 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Elétrica.

Inclui bibliografia

1. Engenharia Elétrica – Teses. 2. Aprendizagem profunda;. 3. Redes neurais convolucionais;. 4. Transferência de aprendizado;. 5. Segmentação semântica;. 6. Agricultura de precisão;. I. Caarls, Wouter. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Elétrica. III. Título.

CDD: 621.3

To my parents and girlfriend for their support,
love and encouragement.

Acknowledgments

I would like to thank my advisor, Prof. Wouter Caarls, for his wise and patient guidance and his enthusiasm in this project.

Then I wish to thank Prof. Antonio C. Leite for his interest and support in this project.

I thank Prof. Pål J. From and Diku[1] for the financial support for the trips to Norway where I participated in the "Vision-based control with applications to robotic systems" project which was essential for the development of the research theme.

I thank CNPq and PUC-Rio for the partial financial support of this work.

I would also like to thank Umoe BioEnergy and Embrapa Agrobiologia for the opportunity to visit their agricultural fields for experimental data collection.

I would like to thank the researchers from the Applied Computational Intelligence Laboratory and the opportunity to use their infrastructure.

Finally, I would like to thank my family for their support, comprehension and encouragement.

[1]Diku - Norwegian Agency for International Cooperation and Quality Enhancement in Higher Education

Abstract

Tenorio, Gabriel Lins; Caarls, Wouter (Advisor). **Applications of Deep Learning for Crop Monitoring: Classification of Crop Type, Health and Maturity**. Rio de Janeiro, 2019. 112p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

Crop efficiency can be improved by continually monitoring their state and making decisions based on their analysis. The data for analysis can be obtained through images sensors and the monitoring process can be automated by using image recognition algorithms with different levels of complexity. Some of the most successful algorithms are related to supervised Deep Learning approaches which use a form of Convolutional Neural Networks (CNNs). In this master's dissertation, we employ supervised deep learning models for classification, regression, object detection, and semantic segmentation in crop monitoring tasks, using image samples obtained through three different levels: Satellites, Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs). Both satellites and UAVs levels involve the use of multispectral images. For the first level, we implement a CNN model based on transfer learning to classify vegetative species. We also improve the transfer learning performance by a newly proposed statistical analysis method. Next, for the second level, we implement a multi-task semantic segmentation algorithm to detect sugarcane crops and infer their state (e.g. crop health and age). The algorithm also detects the surrounding vegetation, being relevant in the search for weeds. In the third level, we implement a Single Shot Multibox detector algorithm to detect tomato clusters. To evaluate the cluster's state, we use two different approaches: an implementation based on image segmentation and a supervised CNN regressor capable of estimating their maturity. In order to quantify the tomato clusters in videos at different maturation stages, we employ a Region of Interest implementation and also a proposed tracking system which uses temporal information. For all the three levels, we present solutions and results that outperform state-of-the art baselines.

Keywords

Deep learning; Convolutional neural networks; Transfer learning; Pixel-wise semantic segmentation; Precision agriculture;

Resumo

Tenorio, Gabriel Lins; Caarls, Wouter. **Aplicações de Aprendizado Profundo no Monitoramento de Culturas: Classificação de Tipo, Saúde e Amadurecimento de Culturas**. Rio de Janeiro, 2019. 112p. Dissertação de Mestrado – Departamento de Engenharia Elétrica, Pontifícia Universidade Católica do Rio de Janeiro.

A eficiência de culturas pode ser aprimorada monitorando-se suas condições de forma contínua e tomando-se decisões baseadas em suas análises. Os dados para análise podem ser obtidos através de sensores de imagens e o processo de monitoramento pode ser automatizado utilizando-se algoritmos de reconhecimento de imagem com diferentes níveis de complexidade. Alguns dos algoritmos de maior êxito estão relacionados a abordagens supervisionadas de aprendizagem profunda (*Deep Learning*) as quais utilizam formas de Redes Neurais de Convolucionais (*CNNs*). Nesta dissertação de mestrado, empregaram-se modelos de aprendizagem profunda supervisionados para classificação, regressão, detecção de objetos e segmentação semântica em tarefas de monitoramento de culturas, utilizando-se amostras de imagens obtidas através de três níveis distintos: Satélites, Veículos Aéreos Não Tripulados (UAVs) e Robôs Terrestres Móveis (MLRs). Ambos satélites e UAVs envolvem o uso de imagens multiespectrais. Para o primeiro nível, implementou-se um modelo CNN baseado em *Transfer Learning* para a classificação de espécies vegetativas. Aprimorou-se o desempenho de aprendizagem do *transfer learning* através de um método de análise estatística recentemente proposto. Na sequência, para o segundo nível, implementou-se um algoritmo segmentação semântica multitarefa para a detecção de lavouras de cana-de-açúcar e identificação de seus estados (por exemplo, saúde e idade da cultura). O algoritmo também detecta a vegetação ao redor das lavouras, sendo relevante na busca por ervas daninhas. No terceiro nível, implementou-se um algoritmo *Single Shot Multibox Detector* para detecção de cachos de tomate. De forma a avaliar o estado dos cachos, utilizaram-se duas abordagens diferentes: uma implementação baseada em segmentação de imagens e uma CNN supervisionada adaptada para cálculos de regressão capaz de estimar a maturação dos cachos de tomate. De forma a quantificar cachos de tomate em vídeos para diferentes estágios de maturação, empregou-se uma implementação de Região de Interesse e propôs-se um sistema de rastreamento o qual utiliza informações temporais. Para todos os três níveis, apresentaram-se soluções e resultados os quais superam as linhas de base do estado da arte.

Palavras-chave

Aprendizagem profunda; Redes neurais convolucionais; Transferência de aprendizado; Segmentação semântica; Agricultura de precisão;

Table of contents

List of figures

List of tables

List of Abreviations

ANN – Artificial Neural Networks
BN – Batch Normalization
BIC – Brain Imaging Center
CCV – Community Core Vision
CNNs – Convolutional Neuronal Networks
CPUs – Central Processing Units
DL – Deep Learning
DSC – Depthwise Separable Convolutions
FPS – Frames Per Second
FT – Fine-Tuning
GCH – Global Color Histogram
GPUs – Graphics Processing Units
IoU – Intersection over the Union
LBSO – Load Balance System Optimization
LR – Learning Rate
MKL-DNN – Math Kernel Library for Deep Neural Networks
MSE – Mean Squared Error
NDVI - Normalized Difference Vegetation Index
PCA – Principal Component Analysis
RANSAC – Random Sample Consensus
ReLU – Rectified Linear Units
RMSE – Root Mean Square Error
RoI – Region of Interest
RP – Region Proposal
SGD – Stochastic Gradient Descent
SIFT – Scale-Invariant Feature Transform
SML – Supervised Machine Learning
SSD – Single Shot Detector
SVM – Support Vector Machine
t-SNE – t-Distributed Stochastic Neighbor Embedding
TL – Transfer Learning
UAVs – Unmanned Aerial Vehicles
UGVs – Unmanned Ground Vehicles

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.

Albert Einstein, *" What Life Means to Einstein".*

1
Introduction

For many decades crop monitoring has been done periodically by farmers, aiming as the main tasks, to identify crop growing problems, health and diseases as well as find other plants that impair the crop development. Usually visual perception is not sufficient to verify the crop state, being necessary the farmer's approach and touch, which takes time in large-scale environments and is impractical in hard-to-reach areas. In such cases, there is the possibility of applying unmanned remote systems that are machines equipped with sensors capable of collecting massive amounts of data from different view points. Some of the unmanned systems use multispectral cameras that help to provide better representations of vegetation areas [2, 3, 4]. A few examples of these systems are Satellites, Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs) which differ in terms of captured image level. The Fig. 1.1 shows an example of sugarcane crop image taken from three different imagery levels.

(a) (b) (c)

Figure 1.1: Example of three different imagery levels for sugarcane crop: (a) Satellites; (b) UAVs; (c) UGVs.

Regarding conventional satellite imagery, UAVs encompass a smaller sampling area, but have higher update speed for scene capture. On the other hand, compared to scenes captured by UGVs, UAVs encompass larger areas and have more agility to capture images. Compared to the other two imagery levels, UGVs imagery provides a higher target resolution due to the proximity of the camera to the target and below the canopy.

In this master's dissertation, we employ the different systems (Satellites, UAVs and UGVs) in, respectively, three monitoring applications: vegetation species, sugarcane crop and tomato clusters. The use of the remote systems

can provide a higher monitoring frequency, resulting in a continuous crop monitoring, which may help in obtaining a more precise and faster crop analysis and diagnosis. The monitoring process can be automated by implementing algorithms that can vary from statistical methods, common image processing methods up to the most advanced supervised learning techniques [5, 6, 7, 8, 9]. As a consequence, farmers can obtain better agricultural crop management and make decisions to increase the crop efficiency, decrease the crop loss and improve the production.

With the current constant improvement in Graphics Processing Units (GPUs) and the development of supervised Deep Learning (DL) architectures, applications involving DL in image recognition have been proven to be very successful [10]. DL approaches that involve raw image data as input make use of Convolutional Neural Networks (CNNs). Some relevant types of supervised models that use convolutional architectures are binary classification, regression, object detection and semantic segmentation. The binary classification gives the probability of a given image to belong to predefined classes (e.g. vegetation species) while the regression learns how to estimate and interpolate single or multiple output values for an input image in a given task (e.g. estimate fruit maturity). Object detection uses regression to locate (through predefined geometric shapes) and binary classification to classify multiple objects in the image. Semantic segmentation is able to divide the image into smaller semantically similar regions, classifying an image at the pixel level.

1.1 Research Theme

The initial ideas of this research theme were based on a project[1] to be developed in the Pontifical Catholic University of Rio de Janeiro. The project is about the development of an agricultural inspection system as well the design and construction an autonomous Unmanned Aerial Vehicle (UAV), fed by solar energy, as a tool for long term environmental monitoring missions. One of the tasks executed by the autonomous vehicle takes into account the image capturing of vegetation areas, which are transmitted in real time to a monitoring central. The information is then processed with the use of computers and algorithms that identify and classify plant species. During the master's degree the UAV was not available, but a public dataset was found and used in the first step of the research project. It provided multispectral vegetation images, captured by a satellite. Then, we followed the classification

[1]VANT autônomo, alimentado por energia solar, para missões de monitoramento de longa duração. Executing Institution: Pontifical Catholic University of Rio de Janeiro – PUC- Rio. FAPERJ Notice Nº 04/2016

task of plant species by using supervised deep learning algorithms (Chapter 2).

Next, we received an invitation to participate in the "Vision-based control with applications to robotic systems" project, that took place in Norway as well in Brazil, where the collaboration project[2] between the Brazilian (PUC-Rio) and Norwegian (NMBU) universities has supported all travel expenses for the three events. During the first participation, the development project of the Norwegians' autonomous modular robots (Thorvald) was presented and a growing need for intelligent monitoring of agricultural fields to accompany this autonomy was observed. We implemented image processing algorithms to identify the health in strawberry leaves using the Normalized Difference Vegetation Index (NDVI) [11] by producing composed resultant images between a pair of infrared and color cameras. During the second visit, we collected the first Dataset, through the use of a UAV, from a sugarcane farm located in Presidente Prudente in São Paulo state in Southeast Brazil where we verified agricultural challenges and proposed solutions based on deep learning (Chapter 3). We also collected new data in a different location at Embrapa's Farm at Rio de Janeiro of Southeast Brazil to perform new tests of the proposed solutions. The third visit made it possible to collect the second Dataset captured on a tomato farm (greenhouse) located in the municipality of Vestfold county in Norway where we applied deep learning algorithms and proposed computational techniques to help to improve the production efficiency (Chapter 4).

1.2 Contributions of this Dissertation

We investigate the possibilities of employing Deep Learning models for image recognition in three different datasets at the three imagery levels, combining existing techniques and practical applications. Additionally, we present different approaches and methodologies for these models, towards crop monitoring applications. We also provide a literature review for the relevant types of supervised models mentioned.

This project is divided into three sub-projects, whose main contributions are shown below. The first sub-project resulted in a paper publication in an international conference[3] while the second and third sub-project are intended to be submitted and published in journals.

[2]UTFORSK Partnership Programme from The Norwegian Centre for International Cooperation in Education (SIU), project number UTF-2016-long-term/10097.

[3]ICAART 2019, "11th International Conference on Agents and Artificial Intelligence", 2019. Available: http://www.icaart.org/?y=2019 [May 15, 2019].

Sub-project 1

- Verify the ability of CNNs to learn how to classify plant species from scarce satellite imagery;
- Propose a statistical dispersion method, based on the interclass standard deviation, to improve transfer learning performance by providing information on the learning limit to be transferred;
- Implement a distributed learning method to increase CNN convergence speed in CPUs.

Sub-project 2

- Examine the learning of DL pixel-wise semantic segmentation algorithms applied to sugarcane crops using images from UAVs.
- Verify how to preprocess different color bands images from a pair of cameras to train the DL model;
- Propose a multi-task configuration for the semantic segmentation algorithm to identify the soil, the surrounding vegetation and the sugarcane crops, inferring their health and age.

Sub-project 3

- Check feasibility of DL object detection algorithms to locate cherry tomato clusters in videos;
- Implement CNN regression models and image analysis segmentation techniques to estimate the fruit's maturity;
- Implement a method based on region of interest and propose a tracking system that uses temporal information to count the number of tomato clusters in image sequences.

1.3 Background

In other to better understand the concepts explained in this project, we recommend some Deep Learning books for beginners and intermediate readers in this area. The first recommendation is available online [12]. The second and third recommendations are, respectively, the classic [13] and the current [14] deep learning books.

1.3.1 Supervised Models

The image processing algorithms implemented in this project involve the use of supervised models. The term supervised learning means that the models are able to learn, through training steps, mathematical functions capable of mapping input-output pairs based on training examples (labelled dataset) [15, 16]. The quality of the function is defined by the model's architecture while their parameters are adjusted (updated) during training by minimizing loss functions to decrease the errors between the output mapped and the output labelled for each input. Before the training process, the labeled data is usually split into training (D_{Tr}), validation (D_{Val}) and test (D_{Ts}). During the training phase, the models are trained with D_{Tr} and evaluated with D_{Val}. After the training, the inference model is obtained which allows to perform predictions on the D_{Ts} and evaluate the model's generalization capacity.

1.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural network architectures, commonly applied to image datasets, that have convolutional layers. Each convolutional layer is filled with parallel filter banks (also called trainable kernels) in which parameters are updated during the training phase. To understand how the convolution process works, first we need to verify how the kernels are applied to an input volume (e.g. RGB image). Each kernel can be represented by a 3x3xN cell grid (matrix 3x3 with N channels), where each value in this cell is trainable and its number of channels is equal to the number of channels of the input volume. Each kernel can be interpreted as a sliding window that, mathematically, performs a weighted multiplication through the entire input volume and its channels, covering all the image pixels and producing an output volume. Another trainable parameter is the bias, a scalar value that adds a constant number to all the pixels in the kernels. Each kernel produces an output volume and each one is stacked, forming the feature maps. In practice, all the kernels sliding windows and the bias are applied in parallel at once, which performs a convolution. This convolution process is illustrated in Fig. 1.2.

In order to learn how to solve a given image-recognition task from a labelled dataset, CNNs use these parallel filter banks and auxiliary operators to filter the images in the search of specific patterns (features). Fig. 1.3 shows an example of CNN.

The trainable filters assist in obtaining models robust to image ad-

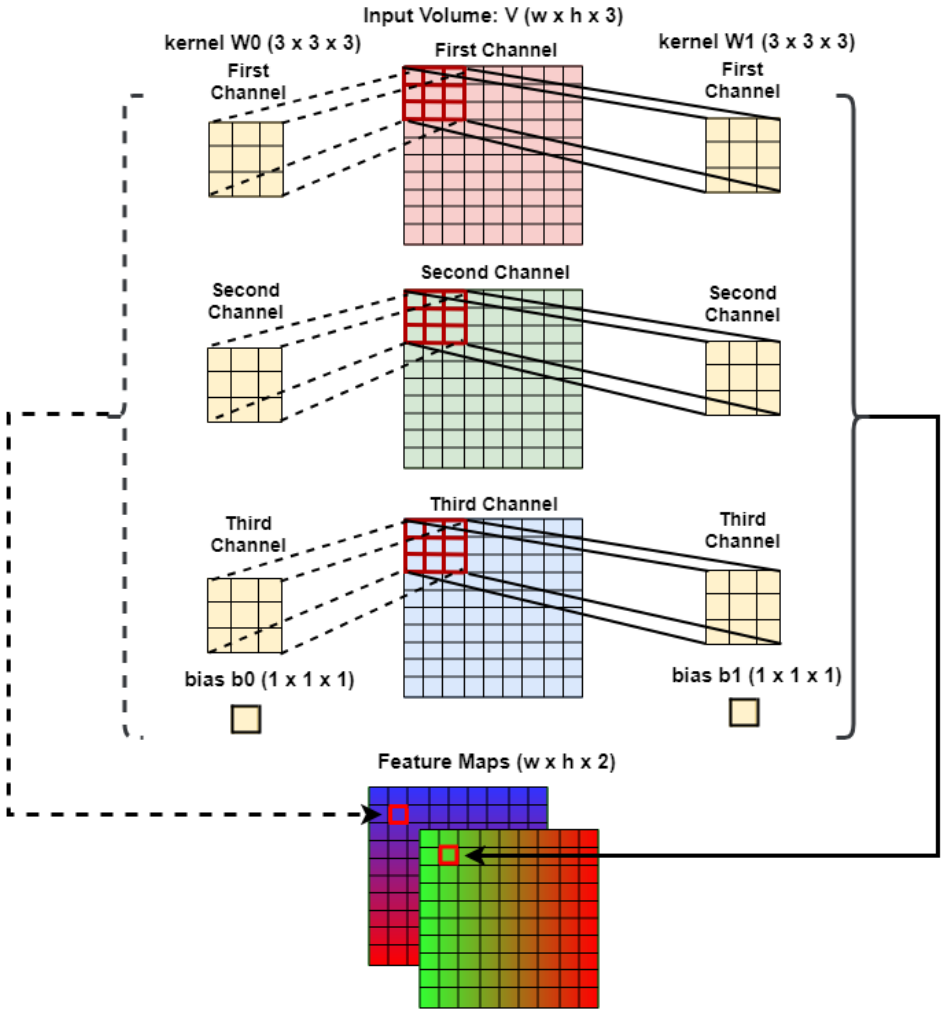


Figure 1.2: Example of a convolution process applied to an input volume V (RGB image), for the first convolutional layer, using two kernels (represented by the W_0 and W_1 yellow cell grids) and their respective bias b_0 and b_1. In this example we assume zero padding, therefore the input dimensions (w and h) do not change after the convolution process. The sliding windows are represented in red. The thick and dotted lines represent, respectively, the production of the output volumes for the $(W_0 + b_0 \cdot J_3) * V$ and $(W_1 + b_1 \cdot J_3) * V$ operations (J_3 is a 3x3x3 matrix of ones). The output volumes are stacked together, producing feature maps that become the new input volume for the next convolutional layer.

verse conditions (e.g. illumination variation, noise and non-homogeneous backgrounds) given as challenges in image processing tasks. Some feature levels examples are the generic, low-level and high-level features. The generic features can be illustrated as color blobs, edges and corners that are present in distinct datasets. The low-level features can be understood as textures that may be different between datasets while the high level features are more abstract and often specific to a dataset.

In common CNNs (e.g. VGG-16 [10]), after the last convolutional layer a

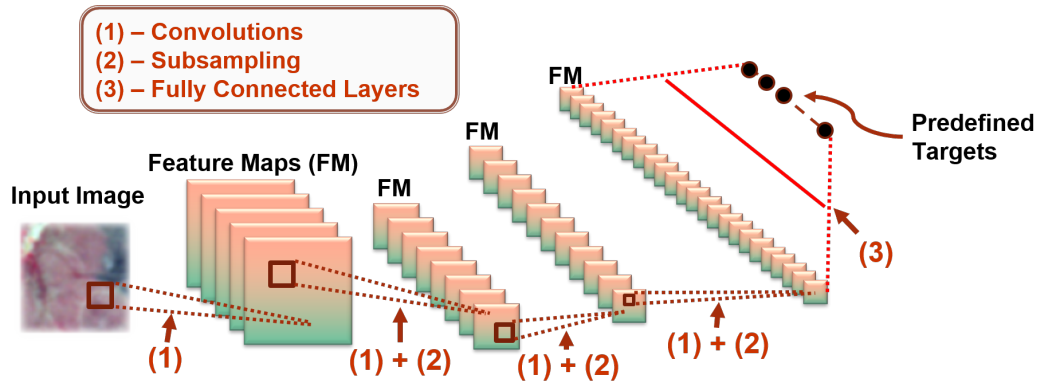


Figure 1.3: Example of an usual Convolutional Neural Network. The convolutions represent the parallel filter banks while the subsampling represent the auxiliary operators. The features obtained at each convolutional layers are are indicated as feature maps.

fully connected layer is attached which role is to combine the feature maps and predict the input data referring to predefined targets. An example of predefined targets are the predefined classes used in classification tasks where each input image is related to a binary number.

1.4 Organization of the Thesis

This master's thesis is organized in three main chapters, where each chapter is related to a sub-project. In the first sub-project (chapter 2), we implement a supervised classifier based on a pre-trained CNN that allows the transfer of existing knowledge to a new task. The objective of the classifier is to identify plant species in a complex and unbalanced multispectral satellite image dataset. We overcome these challenges by employing data augmentation and fine-tuning of the CNN. In order to fine-tune the classifier effectively, the amount of parameters to be transferred and frozen is chosen by a newly proposed statistical method based on the analysis of the CNN convolutional layers.

In the following sub-project (chapter 3), we implement a supervised semantic segmentation algorithm, originally designed to perform pixel-wise classification for a single target. In this sub-project, we obtain multispectral image samples of sugarcane fields, with the use of a pair of cameras embedded in a UAV. We employ computer vision and image analysis algorithms to pre-process the data, obtaining a dataset that helps to improve the learning of the model and consequently improve the segmentation performance. The dataset consists of multi-labelled images containing the information of class (sugarcane, soil and other plants), sugarcane health (healthy, stressed) and the time since

the crop was cut. We adapted the original semantic segmentation algorithm by giving it the ability to learn the three tasks (two classifications and one regression) in a single model, to increase memory efficiency and convergence speed compared to the learning of three independent models.

In the last sub-project (chapter 4), we collect terrestrial tomato cluster videos and capture a certain proportion of snapshots in each video to build the dataset. The samples are labelled in two ways: cluster position coordinates in the image and maturity level of each cluster. In order to detect the clusters in the image, we implement a supervised object detection model. For the maturity level estimate, the solution follows a supervised CNN and an image analysis implementation. The CNN, originally designed for classification, is modified to produce a regression output (continuous value) while the image analysis method employs image segmentation. In order to count the tomato clusters in videos for the different maturation levels, we first combine the detection and evaluations approaches. Next, we employ a region of interest approach that counts each instance whenever the object crosses a vertical line. In order to obtain a result robust to occlusion and capable of distinguishing different clusters, we propose a tracking algorithm that identifies unique clusters based on their coordinates and maturity levels.

2
Improving Transfer Learning Performance: an Application in the Classification of Remote Sensing Data[1]

Keywords: Deep Learning, Convolutional Neural Networks, Transfer Learning, Fine Tuning, Data Augmentation, Distributed learning, Cross Validation, Remote Sensing, Vegetation Monitoring.

Abstract: The present paper aims to train and analyze Convolutional Neural Networks (CNN or ConvNets) capable of classifying plant species of a certain region for applications in an environmental monitoring system. In order to achieve this for a limited training dataset, the samples were expanded with the use of a data generator algorithm. Next, transfer learning and fine tuning methods were applied with pre-trained networks. With the purpose of choosing the best layers to be transferred, a statistical dispersion method was proposed. Through a distributed training method, the training speed and performance for the CNN in CPUs was improved. After tuning the parameters of interest in the resulting network by the cross-validation method, the learning capacity of the network was verified. The obtained results indicate an accuracy of about 97%, which was acquired transferring the pre-trained first seven convolutional layers of the VGG-16 network to a new sixteen-layer convolutional network in which the final training was performed. This represents an improvement over the state of the art, which had an accuracy of 91% on the same dataset.

2.1
Introduction

Vegetation monitoring can be done by farmers to distinguish plants, check planting failures and verify vegetation health and growth. The visual distinction of plants is useful to identify unwanted plants (weed) that deteriorate the health of several species of vegetation [5]. Such monitoring can be difficult when the plantation area is large or when it is fenced by plants. A possible solution is the use of a remote sensing monitoring system using images from satellites.

Some satellites use multispectral sensors which provide images of the visible and invisible spectrum. The reflectance of a plant at a certain wavelength

depends on the flux of radiation that reaches it and on the flux that is reflected. This second variable is conventionally observed in the intensity levels of a plant image in the invisible spectrum of light, as the near infrared spectrum is reflected by the cell structure of plants with high magnitude, varying between different plants [2].

The dataset analyzed in this work was obtained through photo captures taken by the *RapidEye* German satellite system, which provides multispectral data. The dataset contains images from different areas containing four classes of plants: agriculture, arboreal vegetation, herbaceous vegetation and shrubby vegetation, present in the *Serra do Cipo* region in the central area of southern Brazil [17]. For this task, the green, red and near-infrared bands are appropriate for distinguishing the classes of interest [17]. Each image taken by the satellite contains various plant species, making it necessary to consult specialists for separating and classifying them. Thus, a class distribution of the dataset with 1311 multispectral scenes is obtained.

The recognition of the specie's patterns was one of the main difficulties discussed by the original authors regarding the interpretation of the images contained in the dataset, given their complex *intraclass* variance and *interclass* similarity [17]. These issues make the sample preparation and separation into groups costly and limited. As a consequence, there are complex and unbalanced samples so that classification algorithms such as usual *ANN* (Artificial Neural Networks), *SVM* (Support Vector Machine) and *decision-tree* provide unsatisfactory results for this task. However, literature shows that deep learning approaches (i.e. Convolutional Neural Network - *CNN*) and other methods (i.e. data augmentation, transfer learning and fine tuning), have a much better performance in these cases, because they allow to model and train a classifier, (e.g. distinguishing vegetation) using images as inputs, even with scarcity and complexity of data [10]. In this paper, we begin by describing CNN deep learning models and the motivation to use transfer learning and fine-tuning methods. Then, the solution strategy for the classification task is presented along with approaches called data augmentation and cross-validation, to deal with few and unbalanced data. We also show in theory and experimentally how hyperparameters for the model, the methods and the approaches described affect the training performance. In order to train the model with different layer topologies, a statistical dispersion method is proposed to evaluate each convolutive layer of the model and help to choose which layers are the best to be transferred to the fine tune model.

To increase considerably the training speed for the CNN in a *CPU*, parallelism operators and a distributed learning method were used, which

simulates larger minibatches by dividing the data through *workers*. Our experiments are then compared to baselines and state of the art approaches, indicating superior results.

2.2 Theoretical Framework

The convolutional neural network is a type of deep learning architecture that has recently stood out in the image recognition field achieving a very high accuracy [10]. The inputs of a CNN classifier are given by digital images, in the form of tensors, that are brought to feature extractors. Each extractor performs operations, through filters, in parallel to extract features from the images starting from more generic, low-level features and culminating in higher level features that are more specific to the dataset. A second *Neural Network* is conventionally placed after the last convolutional layer to operate as a classifier.

As the complexity of the images increases, there is a need to change CNN hyperparameters. However, as the number of convolutional layers, the filter size and number of CNN filters are increased, the computational cost increases significantly [18]. This effect adds difficulties in experimental research involving real applications, such as those requiring rapid scenario changes [1,2] being necessary to explore the architecture's parallelism capacity.

An approach called Transfer Learning (TL) [19] may decrease the number of required operations allowing the transfer of learning, acquired in one problem, to another problem with similar characteristics. What makes TL more effective is the possibility of using pre-trained networks such as *VGGNet* [20], *GoogleNet* [21], *ResNet* [22] and *AlexNet* [23], which stood out in the challenges of the *Large Scale Visual Recognition Challenge* (*ILSVRC - ImageNet*) for object detection and image recognition [24].

In models that require more specific classification, as in the scope of this paper, there is a need for Fine-Tuning (FT) the model, freezing some layers of the pre-trained networks and constructing convolutional layers on top of them.

[1]Drive.ai, "Building the Brain of Self-Driving Vehicles", 2018. Available: https://www.drive.ai/ [January 29, 2018].

[2]Descartes Labs, "A data refinery, built to understand our planet", 2017. Available: https://www.descarteslabs.com/ [August 15, 2017]

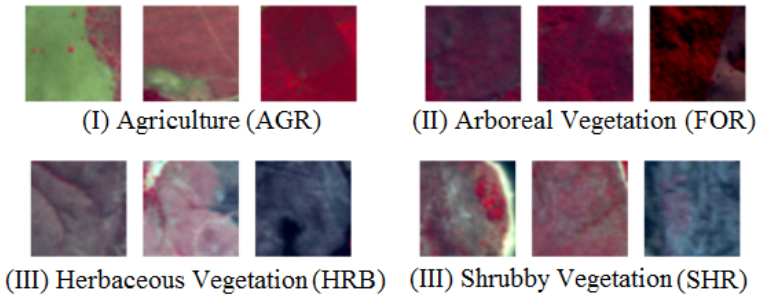


Figure 2.1: Group of four classes: AGR, FOR, HRB and SHR

Table 2.1: Original Species Samples.

Species	**#Samples**	**Proportion (%)**
AGR	47	3.58
FOR	962	73.37
HRB	191	14.57
SHR	111	8.46
Total	1311	100

2.3 Solution Strategy

2.3.1 Preparation of Data and Data Augmentation

The dataset used for the network training [3] was found in the paper by (K. Nogueira et al. 2016) which also uses the artifice of convolutional networks for the classification of four distinct vegetative species, as shown in Fig. 2.1. The resolution of each image is 64 x 64 pixels.

The dataset has unbalanced and scarce samples making it difficult to develop the classification model. Table 2.1 shows the distribution of samples between classes.

Some approaches can be used to train neural networks with unbalanced data avoiding the problem of limited generalization, such as penalizing with higher weights the errors of classification of classes with less samples. Another approach makes it possible to increase the amount of data of each class proportionally, thus balancing the data. In this article, the second approach was chosen to solve the problem of data scarcity. Another advantage of the data increase in a neural network is that it acts as a regularizer, making the model more robust, preventing overfitting [25] and improving the performance of unbalanced models [26].

[3]The dataset is available for download at http://www.patreo.dcc.ufmg.br/2017/11/12/brazilian-cerrado-savanna-scenes-dataset/ [December 6, 2018].

Table 2.2: Balanced classes samples generated using Keras ImageDataGenerator in each new training.

Species	**# Training Samples**	**# Test Samples**
AGR	1770	540
FOR	1770	540
HRB	1770	540
SHR	1770	540
Total	7080	2160

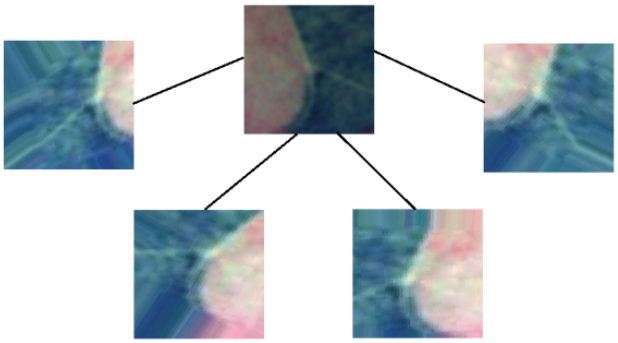


Figure 2.2: Examples of transformations performed by the Keras ImageDataGenerator in a single image.

Through the *ImageDataGenerator* from the python deep learning library *keras*, it becomes possible to generate new images from the dataset with random transformations applied to an image. We used the following transformations: width and height displacement, shear range, zoom, horizontal rotation, and brightness adjustment. Figure 2.2 illustrates four examples of random transformations in a single image. Before increasing the data, the original dataset is divided into training and test sets, respectively, 75% and 25% of the samples. The same sets are used in all experiments. Then, in each new training, the original training set is balanced, proportionally to each class, through data augmentation. After each training, the test set is also expanded proportionally. Table 2.2 illustrates the increase of data using the image generator for the training and test sets.

2.3.2 Pre-trained Network and Transfer Learning

The training of many-layered convolutional networks, based on the random initialization of weights, requires a high computational cost due to the amount of parallel operations that feature extractor filters perform. Using pre-trained networks it is possible to minimize this cost initializing pre-trained weights and bias thereby reaching the convergence of the model much earlier.

Figure 2.3: VGG-16 network architecture. The codes next to the dotted lines indicate the frozen layers in the experiments.

There are several models of pretrained networks such as VGGNet, GoogleNet, ResNet and AlexNet. In this paper, the VGG-16 network was chosen, because it stands out for its uniform and effective architecture for applications involving image classification. The sixteen layers of the VGG-16 network use only 3×3 convolution order and 2×2 order pooling. Convolution is an image filtering process that aims to detect patterns creating feature maps. The pooling process reduces the spatial size of the features discovered by the convolution layers. Fig. 2.3 shows the architecture of the VGG-16 networks [10].

The VGG-16 network is pre-trained using the ImageNet database [27]. This database has about fourteen million high quality natural images with more than a thousand labeled categories, that is, classes with their proper titles.

Transfer learning is a technique that uses pre-trained networks which take the generic features of images, such as color blobs, edges and corners in the first layers. At each subsequent layer, the characteristics taken from the images become more and more specific with the training datasets which can be

obtained, for example, from the imagenet database. After the training step, the classification layers of the pre-trained network (layers 14 to 16) are removed, keeping the previous ones frozen (fixed). The images of the target dataset are executed in this truncated network in order to produce bottleneck features. These features are used to train a new classifier and obtain the prediction of the target dataset classes.

Our target dataset has few samples, besides being very different from the pre-trained network's dataset, so truncating the last layer of that network may not be enough to obtain a satisfactory accuracy. This is because features taken from the layers closest to the output are not useful for the classification of the model. One solution to this problem is fine tuning which allows the lower Y layers of the transfer-learning network to be frozen extracting characteristics from these layers and after the last layer, new convolution and classification layers can be added. A new training of the resulting network can be performed, taking into account that the weights and bias of the Y layers are fixed. Freezing more layers results in a network with lower accuracy in exchange for a training that requires less computational complexity, as the training of a network with less convolutional layers demands less complexity to be calculated. In addition to the freezing of layers, the fine tuning method is also related to the tuning of the hyperparameters of the resulting network.

2.3.3 Cross-Validation

In order to evaluate the learning algorithm and how it responds to the data augmentation, a cross-validation metric is used. The cross validation divides the samples of the increased training set into different training and validation samples by making combinations between them. A specific case of cross-validation is the k-Fold cross-validation method, which divides the samples into k_f subsets at random and without repetition, using all the data in that division. The convolutional neural network model is trained k_f times, where in each training a single subset is selected for the test and $k_f - 1$ subsets for the training. The method is commonly used in Machine Learning applications with 10 folds ($k_f = 10$) aiming at adjusting the generalization of the algorithm [28]. In order to obtain a resulting accuracy, the average of the k_f trainings is calculated.

2.3.4
CNN Layer Statistical Analysis

The output of each CNN layer is a feature map. Interpreting the feature maps in-between the layers may show how well and in which layers the model is learning the specific features of each class. However, they are not trivially interpretable and consequently it is difficult to choose the best layers to be frozen.

Some algorithms for dimensionally reduction (i.e. PCA or t-SNE) can lead us to check whether, in a certain convolutional layer, the features of each class was separated by reducing the dimensionally to two features and drawing a scatter plot [29, 30]. However, in some cases, it is difficult to verify whether the CNN has been able to separate the features or whether the algorithm has been able to reduce the dimension correctly. Another algorithm that can be used to interpret the features maps in CNNs is called DeepResolve [31]. It is based on a gradient-ascent method and does not require inputs by calculating a class-specific optimal "image" H for each class in each layer [20]. This method's output provides helpful information to analyze and decide which layers are important to be frozen.

We propose a simpler statistical analysis of each convolutional layer by calculating the mean of the between-class standard deviation vector, for each layer, which is calculated between the mean feature maps of all classes.

Each three-dimensional feature map matrix is reshaped into a single dimension of vector (feature vector). The standard deviation vector previously mentioned is given by (2-1):

$$\vec{S}_m = \sqrt{\frac{\sum_{n=1}^{N}\left(\vec{\overline{C}}_{n,m} - \vec{\overline{C}}_m\right)^2}{N-1}}\,, \vec{S}_m \in \mathbb{R}_{\geq 0}^{N_m}\,, \tag{2-1}$$

where $\vec{\overline{C}}_{n,m}$ is the mean feature vector between all the images from class n in convolutional layer m and $\vec{\overline{C}}_m$ is the mean vector between the classes in a given layer. Those terms can be calculated by (2-2):

$$\vec{\overline{C}}_{nm} = \frac{\sum_{j=1}^{J_n}\vec{F}_{j_{n,m}}}{J_n}\,, \quad \vec{\overline{C}}_m = \frac{\sum_{n=1}^{N}\vec{\overline{C}}_{n,m}}{N}\,, \quad \left\{\vec{\overline{C}}_{n,m}\,, \vec{\overline{C}}_m\right\} \in \mathbb{R}^{N_m}\,, \tag{2-2}$$

where $\vec{F}_{j_{nm}}$ denotes the feature vector from class n in a layer m of image j and J_n is the number of images in a given class n. Replacing the equations (2-2) in equation (2-1), we calculate the vector $\vec{S}_m$ and the scalar mean M_{Sm} (2-3).

$$M_{Sm} = \bar{\vec{S}}_m\,, \quad M_{Sm} \in \mathbb{R}_{\geq 0} \tag{2-3}$$

The number of classes is four ($N = 4$) and the maximum number of convolutional layers is thirteen ($m = [1, 2, ..13]$). It is expected that the mean of the inter-class standard deviation in each layer (M_{Sm}) increases because higher layers extract more specific features, which should therefore exhibit larger inter-class variance. This variable could tell which is the appropriate convolutional layer that should be frozen and then perform a new training. For example, if the variable decreases considerably in a given layer, this means that the features are getting worse on the new domain (they are too specific to the original domain), and therefore it is not useful to freeze that layer.

2.3.5 Distributed Training Using Large Minibatches

In order to take advantage of the computational power of a multi-core CPU and increase the training speed effectively, a distributed training method was used. The method proposed by [32] simulates large *minibatches* of size kn by dividing the batches of the dataset through k *workers* not compromising, until a certain point, the model's accuracy. In order to maintain the same behavior as a regular minibatch of size n, the method uses a linear scaling rule which consists in multiplying the *learning rate* (η) by the number of workers (k). An assumption is made for this rule to take effect as shown in equation 2-4.

$$\nabla l(x, w_{t+j}) \approx \nabla l(x, w_t), \quad \text{where} \quad j < k\,. \tag{2-4}$$

The first term $\nabla l(x, w_{t+j})$ represents the gradient of the loss function for a sample x and weights w_{t+j} at the training iteration $t + j$. The gradient is used in the minibatch *Stochastic Gradient Descent (SGD)* with a learning rate η and small minibatch of size n [32]. For the large minibatch, the loss is only calculated using the second term $\nabla l(x, w_t)$. With the previous assumption and setting a new learning rate ($\hat{\eta}$) proportional to the number workers ($\hat{\eta} = k\eta$), the SGD updates from small and large minibatch is similar [32]. As an effect, increasing the batch size should not substantially affect the loss function optimization. As described by the authors, the assumption is not true at the beginning of the training when the weights are changing quickly. To solve this problem a *gradual warmup* is used, starting the training from a base learning rate η and increasing this value constantly until it reaches the learning rate $\hat{\eta}$ proportional to k after 5 *epochs* [32]. Additionally, the learning rate is divided by 10 at the 30^{th}, 60^{th} and 80^{th} epochs, similar to [22].

2.4 Experiments

The experiments performed in this work were carried out on two *Intel® Xeon® Platinum 8160* CPUs with 24 *cores* each (96 *threads* in total) and *192GB of RAM* that made possible the application of the distributed training and considerably increase the training speed.

The implemented model was simulated in the Keras framework with Intel® Tensorflow backend that allows the use of dataflow programming. Also, *Intel® MKL-DNN* that accelerates the *Deep Learning framework* on *Intel®* processors (allowing the use of *Load Balance System Optimization*[4] *(LBSO)* parameters) and the open source framework *Horovod* (as a base of the distributed training) were used.

The experiments used *transfer learning* freezing Y layers, denoted *"FL-Y"*, and *fine tuning*. Freezing of the layers below 13 *(FL-13)*, 10 *(FL-10)* and 7 *(FL-7)* were performed as depicted in Fig. 2.3. It is relevant to notice that all the convolutional layers in the *FL-13* experiment are frozen which means that *fine tuning* method has no effect. For all experiments, the *softmax* function was used in the output layer composed of four neurons, which provides the degree of certainty of an input image in relation to each of the four specified classes. The class that contains the highest value is chosen to represent the input image.

The impacts of variations of the hyperparameters of interest, within predefined ranges of values, on the training time and resulting accuracy of the model were analyzed in all experiments. These analyses considered variations in parameters such as dropout, number of epochs and base learning rate. Also, the number of workers (k) and LBSO parameters such as *intra-operation* parallelism (maximum number of threads to run an operator) and *inter-operation* parallelism (maximum number of operators that can be executed in parallel) were evaluated for the model's accuracy. Other parameters such as pooling size and convolution filter size were kept fixed in order to avoid incompatibility with the architecture of the pre-trained networks. For k-fold cross-validation the value of $k_f = 10$ was used, which results in better performance in comparison to lower values of k_f. The SGD optimization algorithm was used for the all the training experiments.

Table 2.3 presents the range of values in which the hyperparameters of interest were evaluated and the optimal values obtained experimentally.

[4]Boosting Deep Learning Training & Inference Performance on Intel® Xeon® and Intel® Xeon Phi™ Processors, 2018. Available: https://software.intel.com/en-us/articles/boosting-deep-learning-training-inference-performance-on-xeon-and-xeon-phi [November 5, 2018]

Table 2.3: Variation of hyperparameters of Interest (keeping the batch size fixed $n = 32$).

Parameter	**Range of Values**		
	Min	Max	**Optimal***
Dropout	0%	70%	50%
# Epochs	35	650	100
Base LR (η)	10^{-5}	10^{-3}	10^{-5}
# Workers (k)	1	12	5
Intra-op	2	48	19
Inter-op	0	4	2
Simul. Batch Size (kn)	32	384	160
Frozen Layers	7	13	7

*The optimal value is the best estimate found for the parameter of interest

Table 2.4: Topologies of the $FL-13$, $FL-10$ and $FL-17$ experiments.

		FL-13	**FL-10**	**FL-7**
Convolutional	Layers	-	3	6
	Filters	-	128x3	512x6
Classification	Layers	2	1	2
	Neurons	512-256	512	512-256
	Dropout (%)	0-0	0.3	0.5-0

The three experiments that provided the best results from the hyparameters adjustments and freezing of the layers below 13 ($FL-13$), 10 ($FL-10$) and 7 ($FL-7$) were selected. As discussed previously, each convolutional layer of the pre-trained network that is not frozen (learning not transferred) must be added in the fine-tuning network. The three best experiments and their topologies are shown in Table 2.4.

2.5 Results and Discussions

In order to evaluate the performance of the model with the different analyzed topologies, the three best experiment configurations of each topology were compared. Additionally, the full training experiment was done. Each experiment was performed ten times, calculating the uncertainty of the results and obtaining more precise accuracy values. Table 2.5 indicates the total simulation time of the training performed, through 10-fold cross-validation, and the resulting accuracy of each experiment. It is important to note that in the original dataset, the proportion of the class with more samples is 73.37%. Considering this observation, it is considered that values of weighted accuracy around 73% are unsatisfactory, because the classifier should be better than chance. To obtain results that are comparable to the original paper, the overall test accuracy of the experiments was also weighted by each class proportion. At each fold of the k-Fold cross-validation method, the model was saved to obtain the class prediction and the normalized confusion matrix on the augmented test set. The diagonal elements of this matrix show the normalized true positive

Table 2.5: Results $FL-13$, $FL-10$, $FL-7$ and $FL-0$.

Experiment	**Training Time***	**Overall Test Accuracy**	**Overall Test Accuracy (Weighted)**
$FL-13$	15 min $\rightarrow$ 4 min	45.1 ± 1.9 %	79.43 ± 1.3 %
$FL-10$	50 min $\rightarrow$ 12 min	81.8 ± 1.4 %	76.7 ± 1.2 %
$FL-7$	183 min $\rightarrow$ 37 min	97.3 ± 0.9 %	97.1 ± 1.0 %
$FL-0$	567 min	55.7 ± 2.1 %	77.6 ± 1.6 %

*The values indicates the decrease ($\rightarrow$) of the training time when using the Intel *MKL-DNN* as opposed to default TensorFlow

predictions of each class which are then weighted by each class proportion and added. This procedure is repeated k_f times and after that the overall test accuracy was calculated. Table 2.6 shows the confusion matrices of experiments *FL-13*, *FL-10* and *FL-7*.

Table 2.6: Normalized confusion matrix *FL-13*, *FL-10* and *FL-7* experiments. The columns indicate the predict class for the test set.

	AGR	FOR	HRB	SHR	AGR	FOR	HRB	SHR	AGR	FOR	HRB	SHR
AGR	0.3	0.26	0.26	0.19	0.85	0.04	0.09	0.02	0.99	0.01	0.00	0.00
FOR	0.01	0.99	0.00	0.00	0.03	0.73	0.05	0.18	0.00	0.97	0.01	0.02
HRB	0.26	0.21	0.23	0.30	0.01	0.00	0.94	0.05	0.00	0.01	0.99	0.01
SHR	0.20	0.29	0.23	0.28	0.01	0.00	0.24	0.75	0.00	0.01	0.03	0.96

In order to evaluate the training time of the experiment that demands most computational power (*FL-7*) varying some hyperparameters for the distributed learning (*intra-op* and k) and keeping the other optimal values fixed (as show in Table 2.3), the Table 2.7 was generated. For all the variations expressed, the model's loss function presents similar behavior and final values.

It is noted from the results presented in Tables 2.5 and 2.6 that the experiment FL-13 has high uncertainty, regarding the achieved accuracy values and it is also unreliable because the most predictions were just in a single class. As noted earlier, the pre-trained network learned from different datasets that are distinct when compared to the dataset employed in this paper. Consequently, only the lower-level features are useful for classifying the specific vegetative species of interest accurately.

Increasing the number of *workers* and *intra-op*, as shown in Table 2.7,

Table 2.7: Training time comparison when varying some hyperparameters using Intel MKL-DNN.

Hyperparameter	Values			
# *Intra-op*	20	48	**19**	12
# *workers (k)*	1	2	**5**	8
Simul. Batch Size (kn)	32	64	**160**	384
Training Time (min)	1598	903	**37**	69

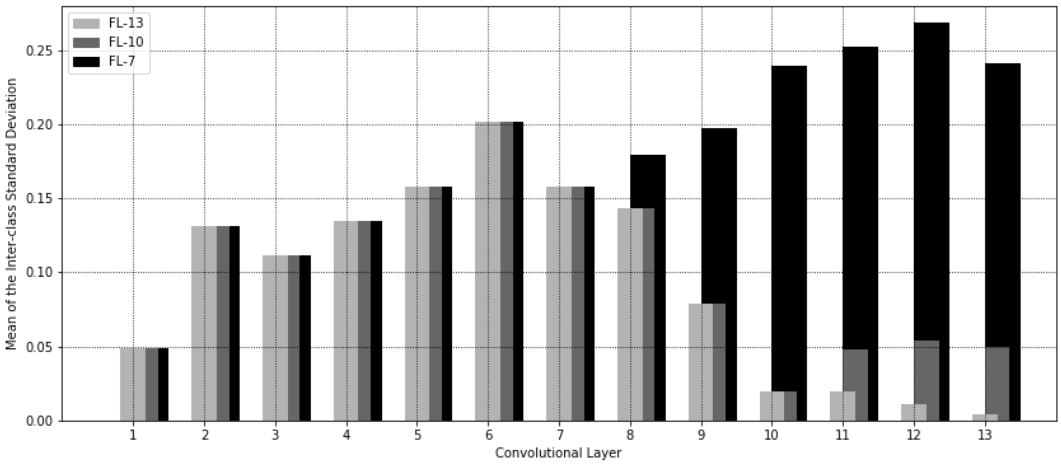


Figure 2.4: Mean of the inter-class standard deviation (M_{S_m}) in each convolutional layer for the *FL-(13,10 and 7)* experiments.

decreases the training time considerably, but at certain point ($k > 8$) the model's performance decreases. Also, a proportionally low value for *intra-op* in relation to k increases the training time and, as a limitation, the multiplication of ($k \cdot$ *Intra-op*) should be less than or equal to the number of total threads ($k \cdot$ *Intra-op* ≤ 96). The best trade-off between performance and training time was obtained when balancing *intra-op* and k values as shown in the highlighted column in Table 2.7.

As discussed and described previously, it is also possible to use a statistical method to find the most suitable layer to be frozen and train the deep learning model. Figure 2.4 shows a chart that contains the mean of the inter-class standard deviation in each layer (M_{S_m}) for each experiment.

Looking at the lightest bars, it can be seen that after the 9^{th} convolutional layer the M_{S_m} value remains low, which confirms that the model can't find useful features that distinguish the classes. So, we may choose to freeze all layers that are part of the 9^{th} layer's max-pooling block and below, leading to the $FL-10$ model.

Alternatively, observing that M_{S_m} has a maximum at the 6^{th} layer, it is logical to freeze until there. Again, fixing the boundary to the max pooling block leads to the $FL-7$ model.

The $FL-10$ and $FL-7$ experiments provided superior results compared to the $FL-13$ experiment that does not use fine tuning. The expressive gain of accuracy of the $FL-7$ and reliability (the true class prediction is more distributed through the classes) of the experiment FL-10 are due to the ability of the convolutional networks, added in these experiments, to learn the more specific features of the dataset used, which was expected for these configurations. In relation to the training time, it can be seen that it tripled

with the addition of three convolution layers ($FL-13$ experiment to $FL-10$) and tripled again by doubling the convolution layers ($FL-10$ to $FL-7$). This proves the expectation of computational complexity attributed to training a network with more convolutional layers.

The superior results can also be observed when looking at the mean interclass standard deviations in Figure 2.4. Both $FL-10$ (dark grey bars) and $FL-7$ (black bars) show higher deviations for the deeper layers, but only $FL-7$ maintains the upward trend of separability.

It is important to note that, for the target dataset, freezing less and less layers results in better results, but in addition to the computational cost, the chances of obtaining poor results are even greater with data augmentation. This is due to, with this decrease, the architecture is increasingly approaching the full training model (i.e., starting from scratch) having a much larger number of parameters to be trained and possibly overfitting the model during the training [10]. In order to confirm this statement, the experiment $FL-0$ (training from scratch) was made and the results are shown in table 2.5. For this experiment, in the training phase, the model's training and validation accuracies presented very high values, but in the testing phase the overall test accuracy value was very low, which indicates an overfitting of the data.

The baseline accuracies for this dataset prediction proposed by the original authors were less than 82.5%, using different techniques as BIC [33], CCV [34], GCH [35] and UNSER [36] as seen in Table 2.8.

Table 2.8: Comparison to baselines and deep learning models for the test set.

Technique	**Weighted Accuracy**	**Technique**	**Weighted Accuracy**
CCV	$80.6 \pm 2.3\%$	BIC	$85.5 \pm 1.4\%$
GCH	$80.1 \pm 2.4\%$	Fine Tuning (original paper)	$90.54 \pm 1.8\%$
UNSER	$80.3 \pm 0.2\%$	**Fine Tuning *(FL-7)***	$97.1 \pm 1.1\%$

The best accuracy obtained by the authors of the article used as inspiration was $90.5 \pm 1.8\%$. In the mentioned paper, the AlexNet pre-trained network architecture with fine tuning and layer freezing was used, but without data augmentation [17]. The architecture of the AlexNet network is different from the one used in the present article having a smaller number of parameters and greater complexity of operations. The results obtained with VGG-16 and the auxiliary methods specified above have shown to be promising (accuracy about 97%) in relation to those with differentiated architecture. The possible differences in results are related to the use of data augmentation and the VGG-16 network in having fixed parameters such as pooling size and convolution filter size. In this way, with the change of the other hyperparameters, the im-

pacts of these variations are more effectively realized, resulting in a satisfactory fine-tuning.

2.6 Conclusions and Future Work

This paper used methods that have helped convolutional networks to learn more effectively the specific characteristics of vegetative species groups. The data augmentation method was essential in achieving effective accuracy by balancing the data to prevent overfitting, while transfer learning accelerated the training process of the network, smartly, by skipping training steps. The fine-tuning approach enabled to truncate any layers of the transfer-learning network and the insertion of convolutional layers, to distinguish the classes with higher accuracy. The statistical analysis proposed helps to choose which layers should be frozen avoiding unnecessary extra experimental tests for the correct choice.

The distributed learning (training the model by dividing the dataset between workers) and the tuning of the parallel operators (LSBO parameters) have shown the possibilities to train a convolutional network in a CPU with high training speed. It is relevant to notice that the maximum number of workers that resulted in good performance is limited, perhaps due to the small dataset.

The final result which indicates an accuracy of about 97% is relevant for applications involving remote sensing for the classification of vegetation species whose images are derived from satellites.

The classification model implemented may not work well if the camera is not multispectral, being an essential equipment for the plant classification task. Also, for satisfactory training and inference results, the dataset must be divided into small tiles to reduce the number of classes present in each image.

The theoretical and experimental study of solutions for implementing a classifier using unbalanced data have great importance for future work, since most environmental monitoring applications rely on disproportionate class data. It is intended to apply the concepts and experiences learned in new datasets with more classes and more data. Also, for future work, pixel-wise semantic segmentation deep learning models [9, 37, 38] may be used to classify the plants species which makes it possible to classify whole images containing multiple classes at the same time and without being necessary to crop them into small tiles.

Acknowledgement

This work was partially funded by a Masters Scholarship supported by the National Council for Scientific and Technological Development (CNPq) at the Pontifical University Catholic of Rio de Janeiro, Brazil.

3
Sugarcane Monitoring using Pixel-Wise Semantic Segmentation

Keywords: Deep Learning; Convolutional Neural Networks; Pixel-wise semantic segmentation; Remote Sensing; Vegetation Monitoring; Image Analysis; Computer Vision

Abstract: This paper consists in developing a vegetation monitoring system capable of detecting sugarcane crops, estimating their health levels and their age range through pixel-wise semantic segmentation. Also, the system detects soil and other species of plants which may include invasive species (weeds). The system formulation involves obtaining sugarcane samples using remote sensing images captured by a pair of cameras embedded in an Unmanned Aerial Vehicle. In order to preprocess the data, Image Analysis (e.g., image segmentation) and Computer Vision (e.g., camera calibration, obtaining corresponding features and image transformations) strategies were applied. The data obtained through the methods described are then combined to produce samples that are intended to improve the semantic segmentation. The research project also involves sample labeling to train a Convolutional Neural Network (CNN) used as a baseline and the main Fully-CNN encoder-decoder segmentation algorithm. To address the multiple target segmentation problem, we propose a multi-task configuration for the Fully-CNN model. After training the models, reaching satisfactory convergence with auxiliary techniques (e.g., data augmentation and early stopping), we verified that the main algorithm outperformed the baseline showing accuracy values between 86% and 94% and intersection over union between 0.74 and 0.81 for all the classification tasks. For the regression task evaluation, it indicates a Root Mean Square Error of 1.22. Using images obtained from a different equipment in a different location, we verify the visual performance of the classification algorithm.

3.1 Introduction

Traditional vegetation monitoring is based on manual acquisition of terrestrial data at predefined time intervals. The numerical data obtained is generally analyzed by statistical algorithms aiming to identify the vegetation change, evaluate the agricultural efficiency and identify its determinant factors [39, 40]. The terrestrial data to be obtained is costly and demands time, despite the sparse availability of data which negatively affects the statistical results in large areas [39]. With the current ease of accessing remote sensing equipment (e.g. Satellites and Unmanned Aerial Vehicles), the data, in the format of images, can be quickly collected, allowing not only real-time, but also a more robust vegetation analysis. The aerial high-resolution images obtained make it possible for the analysis to more precisely cover large regions. This data analysis can help farmers to obtain better agricultural crop management and make decisions to improve its efficiency.

Remote sensing cameras embedded on specific Unmanned Aerial Vehicles (UAVs) provide multispectral images which contains both visible and invisible spectral bands that can be combined to produce better representations for identifying the vegetation properties. The Normalized Difference Vegetation Index (NDVI) is a representation that consist of the composition of the images taken in both spectra, where the visible captures the absorption and the invisible the reflection of the light incident on the vegetation [2, 3]. This index can be used to identify vegetation, estimate vegetation health and biomass and locate diseases in the vegetation for prevention or early treatment [3, 4]. Some disadvantages of using the standalone NDVI are related to its susceptibility to variables such as plant type, soil moisure, and amount of rainfall, leading to possibly noisy results being necessary to employ algorithms that deal with uncertainty [41, 42, 43].

Besides the image processing involving NDVI for vegetation monitoring, current applications such as detecting plant diseases and identifying unwanted plants (weeds) generally use Supervised Machine Learning (SML) techniques, helping to enhance the crop efficiency [6, 8]. These plants when growing outside their native habitat can cause damage to the local vegetation and impair its development. Multispectral images with the aid of SML algorithms can simplify the identification of weeds growth by segmenting the image for later application of localized herbicides [8]. In the field of machine learning, the algorithms that stand out in image recognition are deep learning models that uses Convolutional Neural Networks (CNNs) which provide outstanding results in countless distinct tasks [10]. In vegetation monitoring, these deep learning

algorithms use trainable filters applied to images from real environments or NDVI images to mitigate the noise and remove information that it considers irrelevant to solve a given task. Therefore, the monitoring can be done with a higher precision.

In this research, we collected aerial image samples from a sugarcane farm located in Presidente Prudente in São Paulo state in Southeast Brazil. In some Brazilian regions, the sugarcane vegetation is present in extensive areas and the sugarcane crop has a relevant importance in the renewable and clean production of ethanol energy, therefore improving the crop efficiency can bring benefits to the environment and the farm itself [40, 44]. The data were obtained by a pair of color cameras embedded in an UAV, providing RGB and RGB + IR images. We begin by describing the sugarcane dataset and the collected crop information for different areas. Then, we present computer vision methods to calibrate the cameras, find the correspondences between the pair of images and apply transformations to combine them. In order to pre-process the images aiming to obtain label data, a method to correct the white balance between sequential images is proposed. Next we describe the desired targets for each region in the image, such as vegetation classification, their health state and their age range. After labeling the data (creating image masks), we divide each image into smaller parts to deal with variable aspect ratio caused by the image pair matching. We implemented supervised algorithms, such as, a CNN as a baseline and an Encoder-Decoder segmentation architecture as the main model, proposing a configuration to achieve multiple task goals for training and inference. Our experiments take into account the training of the models using the combined RGB and RGB + IR as well as the individual RGB images. We also employ data augmentation and early stop techniques to improve the final results, avoiding overfitting.

3.2 Background

Image segmentation is an image analysis technique applied to digital images to simplify their representation by splitting the image into different coherent regions (e.g. regions that are related by brightness, color and texture), aiming to ease its analysis [45]. A fundamental example is color segmentation which uses logical operators (thresholds) to divide the image into sets of pixels, based on the pixel intensities. The thresholds can be chosen manually by analyzing the image histogram and verifying the behavior of the processed image or using statistical algorithms, such as Otsu's method to find optimal thresholds values [46]. In this paper, this technique is used as a initial

point to obtain the labels for the supervised models implemented. The main drawback of color segmentation is the susceptibility to factors as variations in luminosity and the noise due to non-homogeneous backgrounds which are common in images captured in real environments. Some unsupervised segmentation methods are robust to these factors, such as the region-based and edge-based watershed transform, the texture-based Gabor filters and the diffusion based feature space [7, 47, 48].

A current way to segment images that has been highlighted in the field of image recognition is semantic image segmentation. Differently from the segmentation methods previously described, semantic segmentation is done through supervised algorithms to split the image into regions that share semantic characteristics. In order to achieve this, these algorithms are firstly trained with labelled image data (masks based on predefined classes), then an inference model is obtained to classify each pixel in the image. The supervised algorithms that provide the most relevant segmentation results involve deep learning CNNs models with different architectures, such as SegNet, U-Net and Deep Lab which differ superficially in relation to complex operations, speed in terms of inference and robustness to image adverse conditions as well handling multiple classes [9, 37, 38]. In this paper, we opted to use the SegNet architecture that is described in details in Sec.3.4.1.

3.3 Dataset

In this section we begin by describing the data acquisition system for obtaining image samples at different planting areas. Next, we present approaches to preprocess and label the data to obtain the resultant dataset to be used by the supervised algorithms.

3.3.1 Image Data Acquisition Setup

With the purpose of obtaining the dataset samples, two color cameras were used. The first camera is a DJI Phantom 3 camera (C_{RGB}) that provides RGB images (I_{RGB}). The second camera, a modified Canon Powershot S ($C_{(\mathrm{RGB+IR})}$), produces RGB + IR images ($I_{(\mathrm{RGB+IR})}$) due to removal of the infrared low-pass filter. The cameras were attached to the Phantom 3 UAV at an angle perpendicular to the ground.

3.3.2
Study of Crop Field in Different Areas

The image samples collected by the UAV's embedded cameras comprise three sugarcane vegetation areas. For each area, a specialist provided the respective crop information. Additionally, the GPS coordinates of each image captured were saved to remotely identify the Geo-position of each crop condition. The coordinate of each image can be visualized in geobrowsers (e.g. Google Earth). Fig.3.1 and Fig.3.2 show the GPS coordinates of all the samples collected, visualised in Google Earth geobrowser.

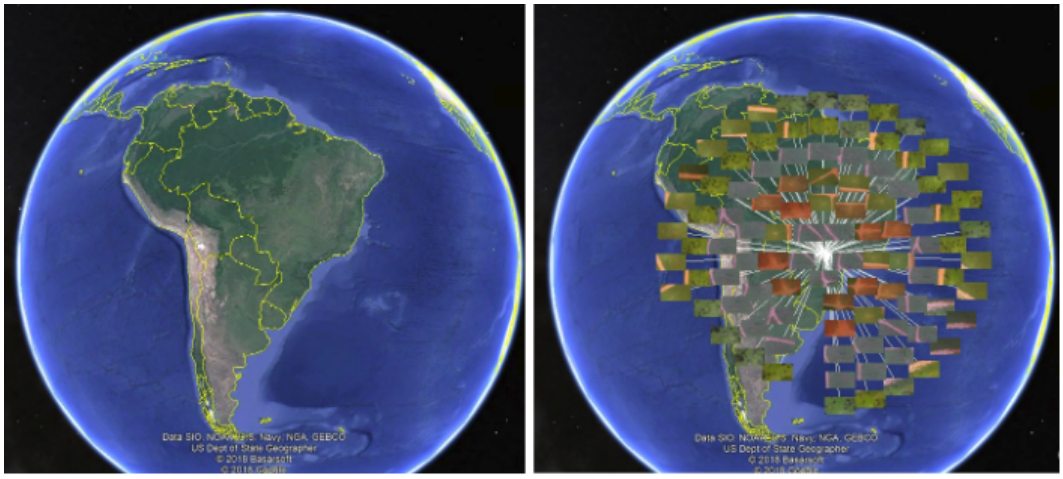


Figure 3.1: Samples location using the camera's Global Positioning System.

Figure 3.2: Samples location in the three areas (zoomed in) using the camera's Global Positioning System. Each area contains a set of images captured.

3.3.2.1
Area 1

The information obtained for the first area indicates low and medium height sugarcane crops. Additionally, the crops in this area are considered healthy. The Fig.3.3 shows the samples location in the first area.

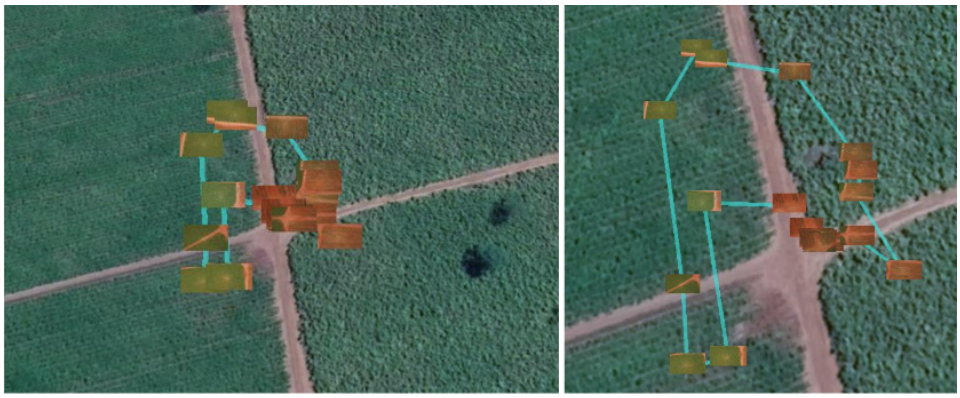


Figure 3.3: GPS coordinates for the first area. The left and right parts of the images are, respectively, the low and medium height crops.

3.3.2.2 Area 2

The vegetation in this area contains stressed low and medium height sugarcane crops. Other species of plants may also be observed in the samples. The present crops were being destroyed by *Cigarrinha* plague and they also have red streak and black rot diseases. The Fig.3.4 shows the samples location collected in this area.

Figure 3.4: GPS coordinates for the second area. The sugarcane health is not clearly visible, since the images captured by the satellites represent a different time.

3.3.2.3 Area 3

The third area contains both stressed and healthy medium height crops. Also, other plant species are present in greater quantity. The Fig.3.5 shows the samples location in the last area.

Figure 3.5: GPS coordinates for the third area. The left and right part of the images shows, respectively, the sugarcane crops and the other plant species

3.3.3 Image preprocessing and Labeling

In order to prepare the dataset, the image samples were processed with computer vision and image analysis algorithms. Initially, the cameras were calibrated to correct the image distortion and improve the image registration. Also, for the color images, the white balance and exposure were adjusted to provide coherence between images from the same area. After the image registration, the red band images of each camera were concatenated, producing two-band resultant images ($I_{\mathrm{R(R+IR)}}$). The initial labeling was done through color segmentation using a proposed NDVI variation denoted NDVI^{l} that is produced by combining the two image bands. These masks were improved using a manual segmentation annotation tool that is fed with the $I_{\mathrm{R(R+IR)}}$ images and the initial masks. The Fig. 3.6 shows the processing flowchart of the image preprocessing and labeling.

3.3.3.1 Geometric Camera Calibration

When using different cameras, the images obtained at the same time are geometrically dissimilar, mainly due to the lens distortion that may hinder the joint processing. Camera calibration uses mathematical equations to estimate the camera's sensor and lens parameters [49, 50]. The equation's constants are the intrinsic, extrinsic and distortion coefficients that are obtained after matching world point coordinates to their corresponding image points by sampling multiple images and fitting the points to the equations [51]. These samples are obtained by taking pictures of a flat calibration pattern (e.g. chessboard) in multiple perspectives. One of the benefits of camera calibration is to use the lens distortion parameters to mitigate the radial and tangential

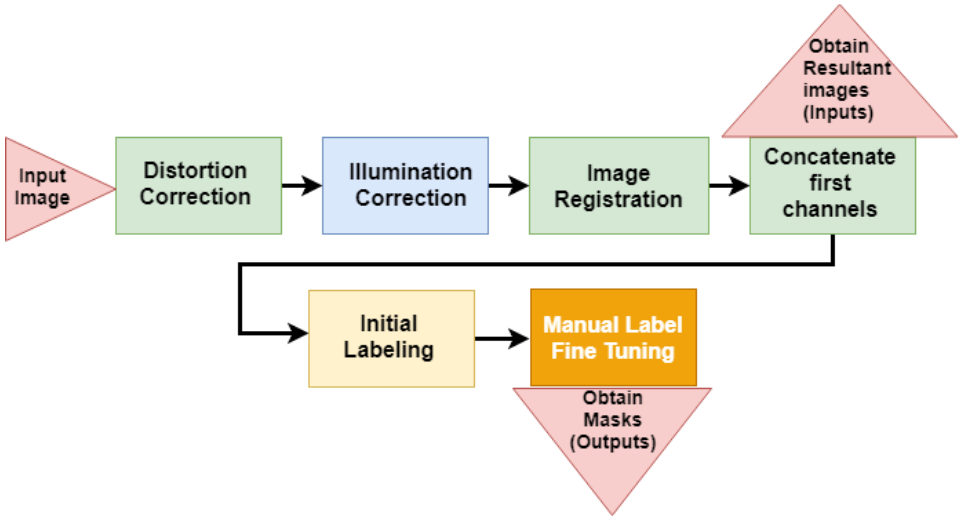


Figure 3.6: Image processing and labeling flow chart. The green color indicates the processes applied to both cameras, while the blue color to the *RGB* images, the yellow to the NDVI′ and the orange color to the R(R + IR) images.

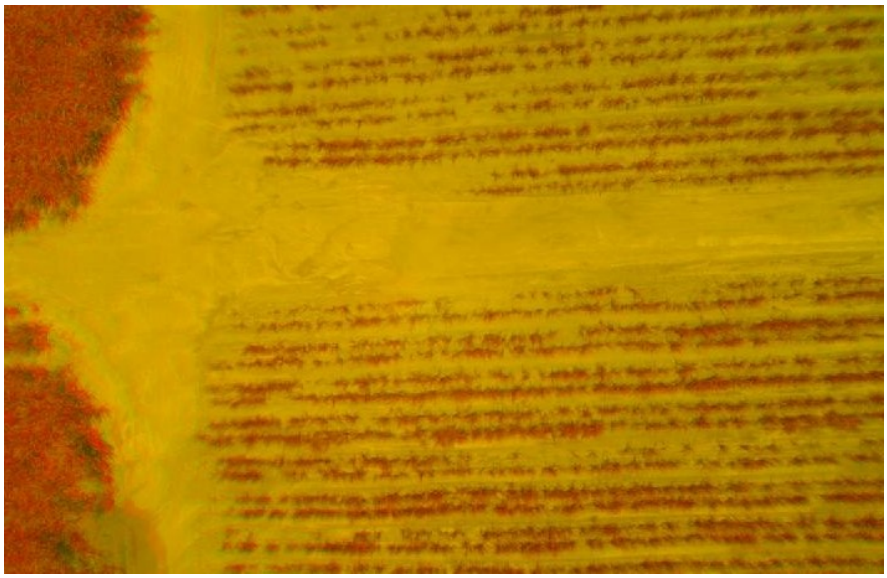
distortion in the images, obtaining rectified images. The color camera used in this paper has a wide-angle lens, being necessary to use more radial distortion coefficients to rectify the produced image. Fig. 3.7 shows an example of the differences between the image overlap (produced by matching and concatenating the red band images of each camera) when aligning the distorted and rectified image pairs.

As expected, there is no distortion at the center of both overlapping images (a,b), but the radial distortion is highly noticeable at the borders of the image (a). It is also noticeable that due to the misalignment, the image (a) is blurred at various regions outside the center. Therefore, the image rectification is essential for joint processing.

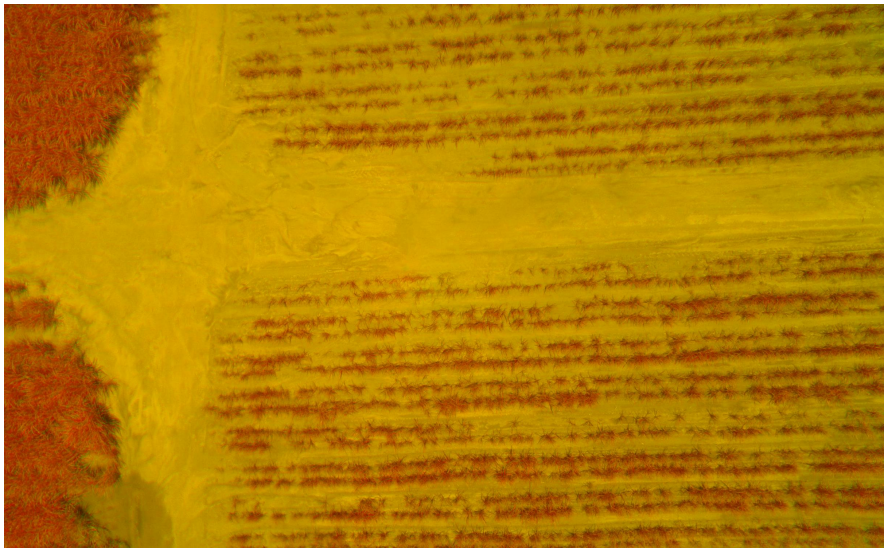
After rectifying all images from both cameras (obtaining $\hat{I}_{(\mathrm{RGB+IR})}$ and $\hat{I}_{\mathrm{RGB}}$), the joint processing was performed efficiently.

3.3.3.2 Image Matching and Alignment

In order to obtain the registration between the rectified images of both image pairs ($\hat{I}_{(\mathrm{RGB+IR})}$ and $\hat{I}_{\mathrm{RGB}}$), firstly we use the feature-based SIFT matching algorithm to find the corresponding points between the image pairs and the RANSAC algorithm to filter the mismatched points [52, 53]. The SIFT algorithm stand out for its robustness to illumination changes, rotation, translation and scaling that exist between the image pairs. After finding the correspondences, a homography transformation matrix is estimated to map each pixel from $\hat{I}_{(\mathrm{RGB+IR})}$ to $\hat{I}_{\mathrm{RGB}}$ in order to align the image pairs (obtaining



(a)



(b)

Figure 3.7: Example of the differences between the image overlap when aligning the distorted (a) and rectified (b) image pairs. The radial distortion is especially noticeable in the bad alignment in the camera (a)

$\hat{I}^T_{(\mathrm{RGB+IR})}$) and also decrease the difference in depths and perspective [54]. Then, for each pair, the first channel is concatenated ($\hat{I}_{\mathrm{R}}$ and $\hat{I}^T_{(\mathrm{R+IR})}$) to produce a two-channel resultant image $I_{R(R+IR)}$. The Fig.3.8 and 3.9 show an example of, respectively, the images matching and the resultant image generated from the transformation and concatenation.

It is important to highlight that the regions outside the intersection between the pairs should be discarded after the registration, as exemplified in Fig. 3.10.

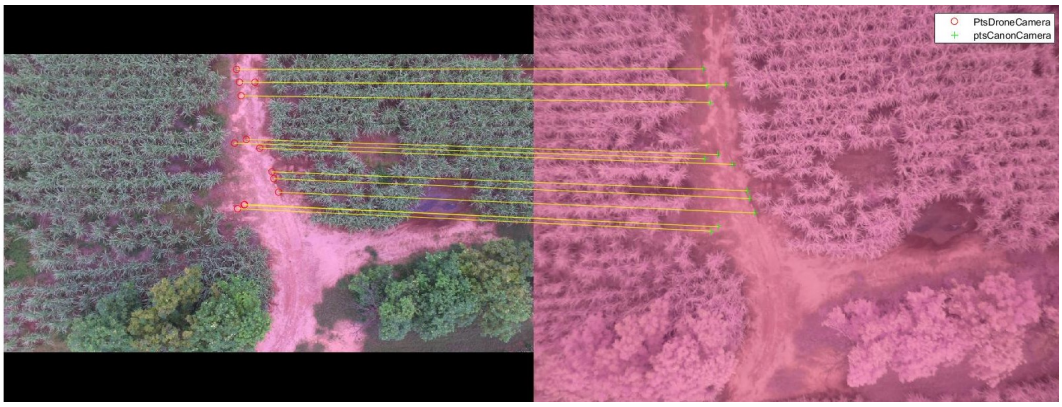


Figure 3.8: Example of image matching between RGB and RGB+IR images using the SIFT and RANSAC algorithms.

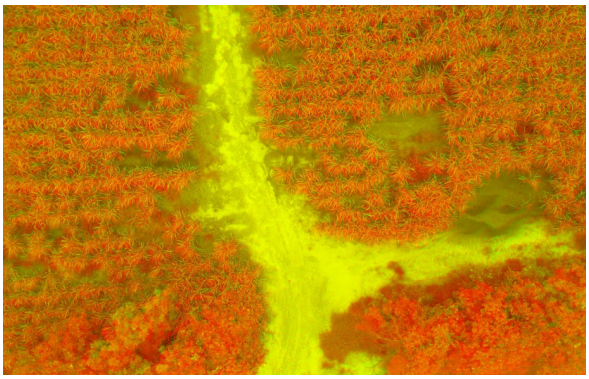
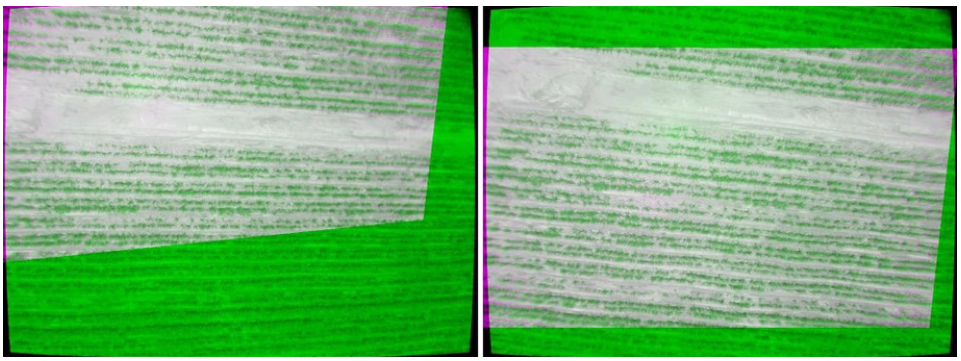


Figure 3.9: Resulting image after applying a transformation matrix to the $\hat{I}_{(\mathrm{RGB+IR})}$, obtaining $\hat{I}^{T}_{(\mathrm{RGB+IR})}$ and concatenating the $\hat{I}_{\mathrm{RGB}}$ and $\hat{I}^{T}_{(\mathrm{RGB+IR})}$ first channels (obtaining a $I_{\mathrm{R(R+IR)}}$ image).



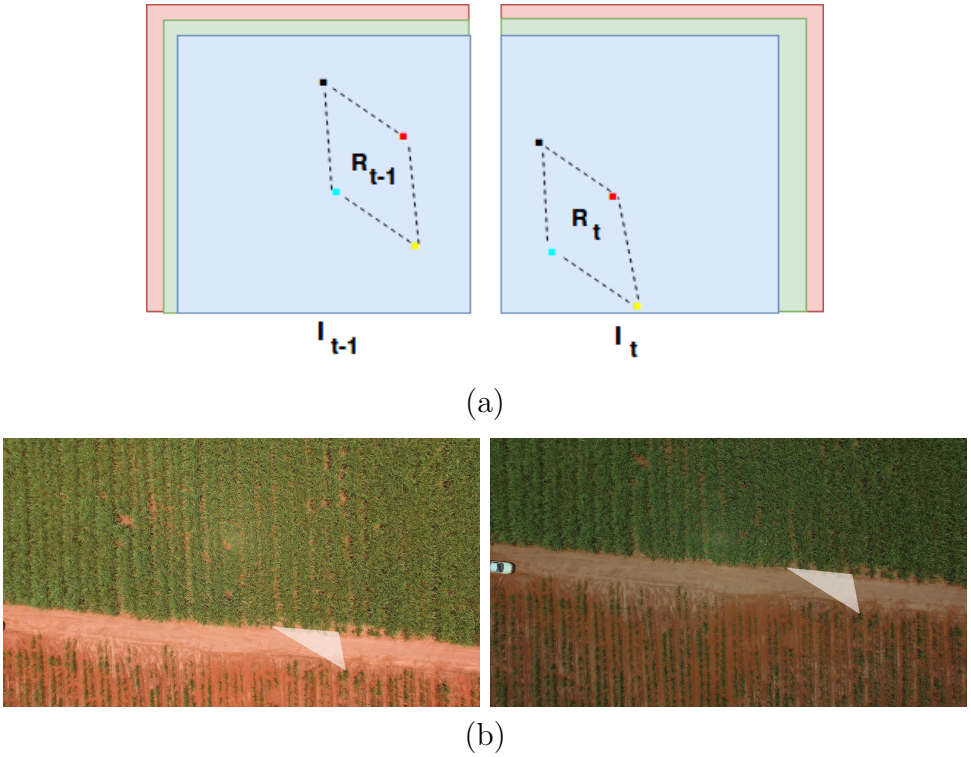
(a) (b)

Figure 3.10: Two examples of image alignment after registration (a) and (b). The regions outside the intersection (transparent green) are removed by fitting the regions of intersection in a crop rectangle tool.

3.3.3.3 Illumination Correction

The samples obtained with the color camera were affected by a varying exposure and different white balances. These effects results in images with

color and intensity inconsistencies which may interfere in the crop health identification. In order to solve this problem, we implemented an algorithm to mitigate the color and intensity offsets between sequential rectified images. In each of the three experimental areas, a sequence of images is followed and the previous image I_{t-1} in the sequence has some similar regions with the current image I_t. The similar regions between the images in the sequence should have approximately the same pixel intensity mean for all channels. We take advantage of the sequences to use the SIFT and RANSAC algorithms to find four pairs of correspondences with the best matches between I_{t-1} and I_t. Then, for each image pair, the algorithm uses the corresponding points to find polygonal regions R_{t-1} and R_t (sub-images) as shown in Fig.3.11.



(a)

(b)

Figure 3.11: Illustration of the algorithm proposed to correct white balance. The first left and the right RGB images (a) represent, respectively, the previous and the current image in a sequence. The color points shown in the figure are examples of the best four pair of correspondences between the images used to calculate the polygonal regions. The second left and right images (b) represent an example of the illustration with real images.

Next, the algorithm calculates the mean of the pixels intensity of each channel inside the regions ($\overline{R_{t-1}}$ and $\overline{R_t}$) to find constants of proportionality between the sub-images. The constants of all channels ($\vec{k}_{(RGB)}$) are calculated using

$$\vec{k}_{(RGB)} = \frac{\overline{R_{t-1}}}{\overline{R_t}}, \quad \vec{k}_{(RGB)} \in \mathbb{R}^3_{\geq 0} \tag{3-1}$$

After obtaining $\vec{k}_{(RGB)}$, the current image channels in the sequence are multiplied by the constants, balancing the pixel intensity and colors. The process is repeated for the whole image sequences.

3.3.3.4 Initial Labeling

As mentioned earlier, some advantages of NDVI images are related to the vegetation identification and its health estimation. Using the resultant images $I_{R(R+IR)}$ we calculate the index using a proposed NDVI^l equation variation (Eq.3-3) that produces different results than the NDVI. The difference is due to the inability to separate the Red band to the IR band in the first channel of camera $C_{(\text{RGB+IR})}$. Therefore, when subtracting the first channel of $\hat{I}^T_{(\text{RGB+IR})}$ by the first channel of $\hat{I}_{\text{RGB}}$ results in an impure NIR (NIR^l) as expressed

$$\text{NIR}^l = \hat{I}^T_{(\text{R+IR})} - \hat{I}_{\text{R}}\,, \tag{3-2}$$

$$\text{NDVI}^l = \frac{\text{NIR}^l - \hat{I}_{\text{R}}}{\text{NIR}^l + \hat{I}_{\text{R}}} = \frac{[\hat{I}^T_{(\text{R+IR})} - \hat{I}_{\text{R}}] - \hat{I}_{\text{R}}}{[\hat{I}^T_{(\text{R+IR})} - \hat{I}_{\text{R}}] + \hat{I}_{\text{R}}} = \frac{\hat{I}^T_{(\text{R+IR})} - 2 \cdot \hat{I}_{\text{R}}}{\hat{I}^T_{(\text{R+IR})}}\,, \tag{3-3}$$
$$-2 < \text{NDVI}^l \leq 1$$

The initial labelling is done using a color segmentation algorithm applied to the NDVI^l images. We implemented a script commanded by a horizontal bar to manually choose appropriate segmentation thresholds capable of separating the vegetation to the soil, obtaining the initial masks. In a logical form, the pixels intensities above the threshold are set to 1 corresponding to vegetation and others to 0 corresponding to soil.

3.3.3.5 Manual Label Fine Tuning

The initial masks only provide the labeling for the vegetation and soil, therefore the sugarcane and the other plants are labelled as the same class. Also, as a consequence of the color segmentation method, the initial masks obtained are noisy. In order to separate the sugarcane from the other plants and obtain the final class masks, we fine tune the initial masks using an image annotation tool that can also denoise the masks [55]. The tool provides options during the labeling to, by means of predefined targets, manually draw a polygon around the region of interest or use superpixels for the task. As a result, the pixels intensity values on the initial masks related to the regions of $I_{R(R+IR)}$ containing other plants are replaced with value 2.

Next, the sugarcane crop health masks can be created by initially copying the class masks and setting the other plants and soil regions pixels to 0 as the health information is not relevant for the two classes. Then, deploying the labeling tool, the pixel intensity related to the crop's health masks are replaced with 2 when the region information indicates stressed crop and leaving the remaining sugarcane region pixels equal to 1 which indicates healthy crop. The same process is done to create the time since last cut masks (cuttime masks), but considering a higher number of targets which are then normalized. The table 3.1 illustrates the possible values for the created masks.

Table 3.1: Possible values for the created masks.

Mask Type	Targets	Possible Values
Class	Soil; Sugarcane; Other Plants	Int(0; 1; 2)
Health	X; Healthy; Stressed	Int(0; 1; 2)
Time since last cut (Cuttime)	0 to 15 months	Float(0.0 to 1.0)

3.3.4 Resulting Dataset

The Dataset samples consist of $I_{R(R+IR)}$ high-resolution processed images from the three experimental areas and their respective segmentation masks assigned to class, health and cuttime. As previously expressed, the direct NDVI calculation is not feasible for the data collected, thus we concatenate the $\hat{I}_{\mathrm{R}}$ and $\hat{I}^{T}_{(\mathrm{R+IR})}$ images expecting that the supervised algorithm could deal with this calculation issue.

Due to the $\hat{I}^{T}_{(\mathrm{R+IR})}$ images produced after the rectangular intersection crop (exemplified in Fig.3.10), the concatenated images result in high aspect ratio variation. Therefore, resizing all the images in order to feed the segmentation supervised model with the same shape data, produces poor resolution per image and may worsen the learning performance. To avoid this interference, we cropped the $I_{R(R+IR)}$ images and their respective masks into 400x400 pixels tiles. This division also considerably increases the number of input images and retains the original global resolution. The same approach is done for the binary CNN classifier (baseline), but cropping the images into 64x64 pixels. This division into small tiles is necessary to decrease the interclass relationship and improve the baseline performance. Table 3.2 shows the image dataset produced after cropping the $I_{R(R+IR)}$ into tiles for the segmentation and baseline models, while table 3.3 shows its division into training, validation and test.

Table 3.2: Sugarcane Dataset per area and after cropping for the two models.

Area	**# Processed Image**		**# Images after cropping**	
1	31		**Supervised Model**	**Baseline Model**
2	15		Segmentation (Encoder-Decoder)	Baseline (CNN)
3	18		Tiles 400x400 px	Tiles 64x64 px
Total	**64**		**821**	**20952**

Table 3.3: Sugarcane Dataset for the two models, divided into training, validation and test.

Supervised Model	**# Input Images**	**# Training Images**	**# Val. Images**	**# Test Images**
Segmentation	821	587 (71.50%)	117 (14.25%)	117 (14.25%)
Baseline	20952	14980 (71.50%)	2986 (14.25%)	2986 (14.25%)

3.4 Methodology

The methodology employed consists of using a semantic segmentation algorithm, originally designed for classification, and proposing a multi-task configuration that can handle a regression and multiple classification tasks simultaneously. The proof of concept that inspired this configuration was done through an implemented baseline which is presented in this section. We also present auxiliary techniques used in the model optimization.

3.4.1 Pixel-Wise Semantic Segmentation Algorithm

The *Segmentation Network* (SegNet) is a deep learning supervised algorithm developed for pixel-wise semantic segmentation [9]. Its architecture is efficient in terms of memory for inference which is relevant in real applications. The SegNet is composed of encoder and decoder networks that share information with each other through corresponding encoder and decoder blocks. Besides the shared information, each corresponding encoder and decoder output has the same shape (same width, height and depth), except the last decoder which depth size is the number of pre-defined classes N_c. The encoder network employs the VGG-16 pre-trained CNN as a feature extractor, removing the fully connected layers which drastically reduces the number of trainable parameters (about 9 times) [20]. In each encoder block there are mainly convolutions and max-pooling operators. The convolutions use trainable filter banks to produce feature maps while the max-pooling calculates the features maxima location aiming to produce slightly shift-invariant features. The indices of the maxima location of each encoder block, extracted from the max-pooling operation, are passed to the corresponding decoder block. After the final encoder block, the features maps obtained are non-linearly up-sampled by the first decoder block operator using the corresponding encoder indices, producing sparse

feature maps. Compared to usual bilinear upsampling, the upsampling method used in this architecture is relevant to mitigate the loss of spatial resolution of the feature maps when using many convolutional layers and max-pooling [9]. Similar to the encoder blocks, the decoder blocks uses convolution layers which, in this case, produces dense feature maps from the sparse features. The decoder process is repeated consecutively using the output of the previous decoder block and the max-pooling indices, up to the final block. Right after this block, a Softmax layer is attached with the objective of classifying each pixel individually (i.e. pixel-wise classification) by generating a N_c channels image containing probabilities. Then, the channels indices of the pixels with the highest probability (in the depth dimension of the image) are used to generate the output mask. Fig.3.12 shows the SegNet architecture.

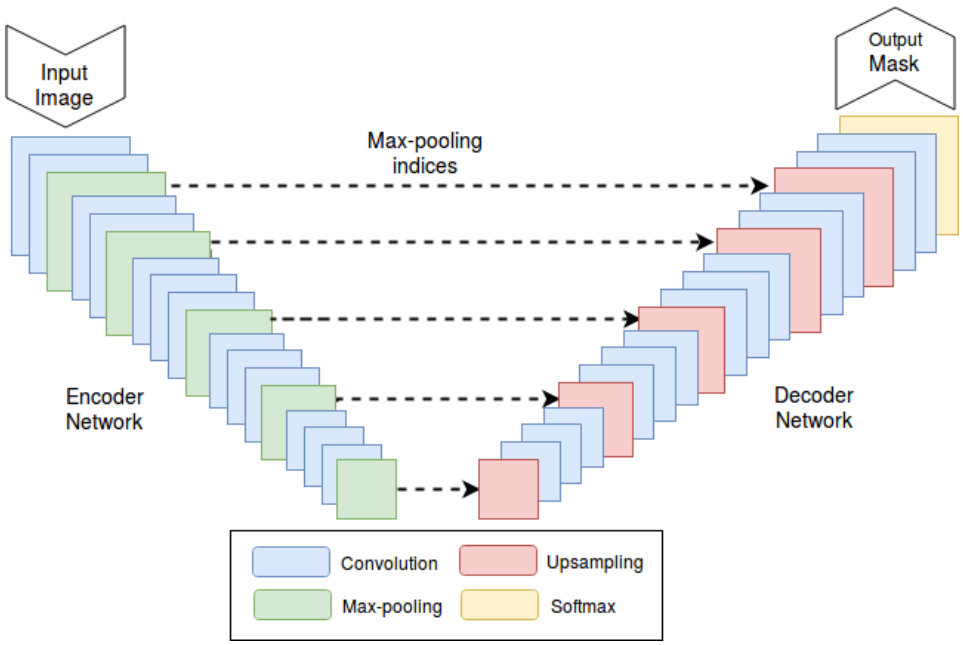


Figure 3.12: SegNet deep learning architecture. The encoder and decoder networks are composed by 5 blocks each, containing 3x3 convolutions and 2x2 max-pooling or upsampling operators. The total number of convolutions in the full architecture is 26, being split in 13 for each network. The dashed lines represent the max-pooling indices calculated in the encoder blocks that are passed to the corresponding decoder blocks.

Two other important operators that follow the convolutions are the activation function ReLU and Batch Normalization (BN) [56]. The ReLU forces the outputs of each convolution to be positive while the BN adds trainable parameters to perform a partial or full normalization on the features maps in each minibatch [57]. As an effect, the BN increases the convergence speed and also acts as a regularizer by passively adding noise to each layer, reducing overfiting during training. Before the first convolution in the encoder network,

the input image is normalized using a Local Response Normalization [58].

Both encoder and decoder networks are trained simultaneously aiming at minimizing the multi-class cross entropy loss function. The encoder network also takes advantage of the pre-trained network to use the pre-trained weights as a starting point and consequently increase the convergence speed.

In order to allow the SegNet to estimate the time since last cut, it needs to perform a regression task. Therefore, we replaced the Softmax layer by a Sigmoid layer and rather than producing N_C channels images in this layer, we only produce single channel images. Also, to complete the regression, the loss function is replaced by the Mean Square Error function.

3.4.2 Baseline Algorithm

As a baseline to accomplish the task, we adopted the VGG-16 convolutional neural network (identical to the encoder network) adapting it to work as a multi-task learning model. The objective to use this adapted model is to, for a given input, produce task-specific outputs (e.g. class, health and cuttime). After the last VGG-16 convolution, task-specific layers are attached (dense layers followed by classification and regression layers) as shown in Fig. 3.13.

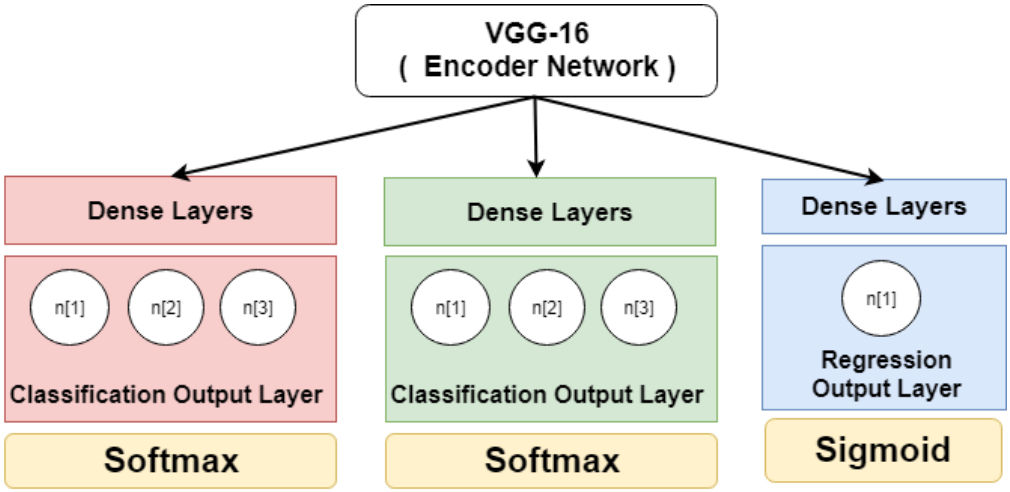


Figure 3.13: VGG-16 CNN adapted as a multi-task learning model. The VGG-16 block represents the original VGG-16 convolutional layers while the following layers are modified. The red and green classification layers perform the classification of, respectively, vegetation class and health. The regression layer (blue) performs the time since last cut estimate.

Each ground truth class mask used to train and evaluate the model for the classification tasks is converted into a single numerical discrete value by taking the value that most repeats in the given mask. Therefore, tile masks

that have more than one class are converted into a single class. Also, each regression ground truth mask is converted by calculating the mean of the numerical continuous values in the given mask.

For the multi-task training configuration, we adopted alternating training in which the trainable parameter updates of the shared layers (VGG-16 convolutional layers) are affected by the three targets, however, the dense, the classification and the regression layers are trained only for the related targets. The trainable parameters are updated aiming in minimizing alternately three loss functions: two Multi-Class Cross Entropy for binary the classification and one Mean Square Error for the regression. At each training step, one of the three loss functions is chosen randomly to be minimized.

It is important to notice that, unlike the SegNet, this model does not provide segmented images as output in the inference stage because the results for a given image tile (64x64 pixels) are three single numbers. In order to produce an approximate effect as the SegNet model (i.e. masks), the single numbers are multiplied by the identity matrix that has the same resolution as the input image. This is feasible, since the tiles are quite small.

3.4.3 SegNet Multi-Task Model

In scientific researches related to semantic segmentation, some multi-task model configurations for the SegNet architecture were proposed. For example, [59] proposed a SegNet cascade configuration for mutually related tasks to produce a distance prediction and a segmentation mask. Another example is [60] that proposed a SegNet configuration in which a shared encoder network and three decoder networks are trained to perform a depth regression, a semantic and an instance segmentation tasks. Both configurations are trained aiming to minimize a specific multi-task loss. Analogously to the baseline implemented and based on [59, 60], we proposed a SegNet configuration to work as a multi-task learning model using the encoder network as the shared layers and three decoders to perform both one regression and two semantic segmentation tasks as shown in Fig.3.14. Our implementation is trained using multiple single-task loss functions, which allows different minibatch sizes to be chosen and adjusted in each task. The idea of our configuration is to encode the images to produce the features common to the tasks and decode them separately for each specify task.

For each input image the model outputs three image masks, but they differ in relation to their pixel data type as the labels discussed in 3.3.3.5. Similar to the baseline, the same loss functions and alternating training are

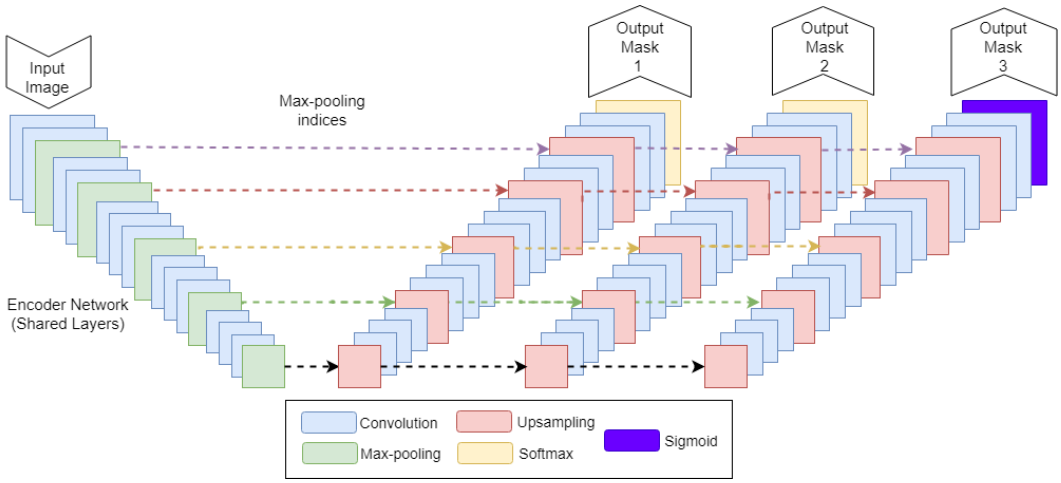


Figure 3.14: SegNet deep learning architecture adapted to work as a multi-task learning model. The encoder network and the max-pooling indices are shared among the three decoders. The output masks, presented in Tab. 3.1, are listed in order (left to right) related to vegetation classification, health classification and time since last cut estimate tasks.

employed in the model. At each alternating training step, a task-specific loss function is randomly chosen to be minimized and the predetermined minibatch size for the current task is selected. The encoder network training is affected by the three tasks while each decoder network is individually affected. During training, the encoder network may eventually learn to extract features giving more emphasis to one task over others. However, due to the nature of the dataset and the pre-trained weights transferred in the encoder network, the relevant features to be extracted by the encoder network are more likely to have an intersection between the different tasks, thus making this approach more efficient. With respect to the decoder network, its learning is focused for each specific task.

Compared to the training of three individual encoder and decoder networks, the proposed configuration (a shared encoder and three individual decoders) uses less memory on training as well in inference and needs less learning iterations due to the considerable reduction of training parameters.

3.4.4 Auxiliary Techniques

To improve the model optimization, we implemented Data Augmentation and Early Stopping techniques.

3.4.4.1 Data Augmentation

This technique allows the application of random transformations in the images and their respective masks during training, aiming to prevent overfiting. In this paper, we use random image rotation and horizontal mirroring.

3.4.4.2 Early Stopping

In order to avoid the increase of generalization error, we use the early stopping method that monitors the loss functions and stops the training before the overfitting point. For our multi-task models, this technique will stop the training using the loss function that has the least decay rate so that a task does not worsen the shared trainable parameters.

3.5 Experiments

The experiments in this paper were performed in Clusters with NVIDIA® Tesla P100™ GPUs with 16GB vRAM memory each, that boosted the training and evaluation speed of the implemented models.

To verify the learning capacity of the Multi-Task algorithms, they were trained separately for both $I_{R(R+\mathrm{IR})}$ and I_{RGB} pre-processed images. Also, to produce comparable results, the I_{RGB} input images use the same regions of intersection mentioned in Fig.3.10 (Section 3.3.3.2). The evaluation metrics for the classification tasks used in this project were the per class pixel-accuracy (P_{Acc}) and the Intersection over the Union (IoU) represented in equations 3-4 and 4-9.

$$P_{\mathrm{Acc}} = \frac{T_P + T_N}{T_P + T_N + F_P + F_N}\,, \qquad P_{\mathrm{Acc}} \in \mathbb{R} : 0 \leq P_{\mathrm{Acc}} \leq 1\,, \tag{3-4}$$

The P_{Acc} represents the percentage of pixels which were correctly predicted for a class in a given image. For the baseline, it represents the true classification predictions for each image.

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union}\,, \qquad IoU \in \mathbb{R} : 0 \leq IoU \leq 1\,, \tag{3-5}$$

The IoU is calculated by dividing the pixels area that intersects and the complementary pixels area between the predicted per-class binary mask and its related ground truth. This metric was only used to evaluate the supervised segmentation model.

To evaluate the regression tasks, only the Root Mean Square Error (RMSE) (expressed in Eq.4-11) was used.

$$\text{RMSE} = N_{\text{months}} \cdot \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(\text{cuttime}_{\text{est}_i} - \text{cuttime}_{\text{gt}_i})^2}\,, \quad \text{RMSE} \in \mathbb{R}_{\geq 0}\,, \quad (3\text{-}6)$$

Where $\text{cuttime}_{\text{est}_i}$ and $\text{cuttime}_{\text{gt}_i}$ represents the pairs of estimated and ground truth continuous values in a given mask. The N_{months} is the multiplicative constant for the number of months (time since last cut variable).

3.6 Results and Discussions

For supervised semantic segmentation models, the number of steps is usually chosen and observed, rather than choosing the number of epochs. With the early stop criteria adopted, the segmentation models finished the training at 22000 steps when training with the $I_{(\text{RGB+IR})}$ images and 26000 steps for I_{RGB} images, both cases using minibatches of size 6 for the classification tasks. The regression task presented a better loss function behavior when using a minibatch of size 4. The training times for the two training cases were about, respectively, 9 and 10 hours.

Tables 3.4 and 3.5 show the generalization of each model (i.e. test set prediction) using the metrics defined in Section 3.5.

To verify the generalization of each model (i.e. test set prediction), the metric equations expressed early were used and the comparative tables 3.4 and 3.5 were built.

Table 3.4: Results when training with the $I_{R(R+\text{IR})}$ images. The Acc and IoU values are the mean values that take into account all the images related to a given class. The RMSE represents the mean of the error prediction.

	Baseline Model						
	Class			**Health**			**Cuttime**
Metrics	*Sugarcane*	*Soil*	*Other Plants*	*Healhy*	*Stressed*	*Other*	
Acc	0.76	0.71	0.79	0.56	0.52	0.60	-
RMSE	-	-	-	-	-	-	5.39
	Main Model						
Acc	0.91	0.92	0.94	0.87	0.92	0.86	-
IoU	0.78	0.80	0.81	0.80	0.79	0.74	-
RMSE	-	-	-	-	-	-	1.22

It can be noticed in table 3.4 that the baseline model was capable of classifying the plants with reasonable accuracy. However the performance for health classification and cuttime estimate was not satisfactory. In table 3.5 the vegetation classification performance for the baseline was similar for sugarcane,

Table 3.5: Results when training with the I_{RGB} images. The Acc and IoU values are the mean values that take into account all the images related to a given class. The RMSE represents the mean of the error prediction.

	Baseline Model						
	Class			**Health**			**Cuttime**
Metrics	*Sugarcane*	*Soil*	*Other Plants*	*Healhy*	*Stressed*	*Other*	
Acc	0.78	0.80	0.51	0.31	0.35	0.41	-
RMSE	-	-	-	-	-	-	12.31
	Main Model						
Acc	0.94	0.93	0.84	0.39	0.40	0.45	-
IoU	0.80	0.80	0.73	0.27	0.22	0.23	-
RMSE	-	-	-	-	-	-	6.23

increased for soil and decreased drastically for other plants. Also, the health classification and cuttime estimate performance got worse.

For the semantic segmentation model, the accuracy per class for the classifications as well the regression results were satisfactory in table 3.4, outperforming the baselines which shows the robustness of the SegNet architecture for the determined tasks. In Tab.3.5, when comparing the trained semantic segmentation models, the vegetation classification was similar for sugarcane and soil, but got worse for other plants. Also, the model trained for health and cuttime when using the I_{RGB} presented poor results due to the indefinite oscillation of the loss functions over the iterations. This effect may indicate the importance of the infrared band for the designated tasks.

In addition to the numerical results, we initially discuss a visual example of the baseline output when inferring each small image tile separately, joining the tiles and forming the original shape segmented image (Fig. 3.15).

A possible source of the low visual performance in Fig.3.15 is that the ground truth information was transformed to provide a single number for each image tile, discarding much information (e.g. when the crop is narrow). Also, differently from the segmentation model, the interclass variance, even reduced by the small tiles division, made the CNN model assign a single class to the input image when more than one class was present.

Next, we verify some semantic segmentation results when inferring each image tile on the test set separately as shown in Figs. 3.16 and 3.17.

In the outputs related to the $I_{R(R+IR)}$ images in Fig. 3.16, the semantic segmentation results were close to the ground truth, but in general the results were affected by small noises. Despite the noises, after joining the tiles together to produce the original shape image these noises may become imperceptible. When analyzing the cuttime visual results for the $I_{R(R+IR)}$ images it can be observed that the regression masks obtained for this task are slightly blurred because the values predicted are continuous and the ground truth provided were integers. The sugarcane growth is not necessarily equal for the whole

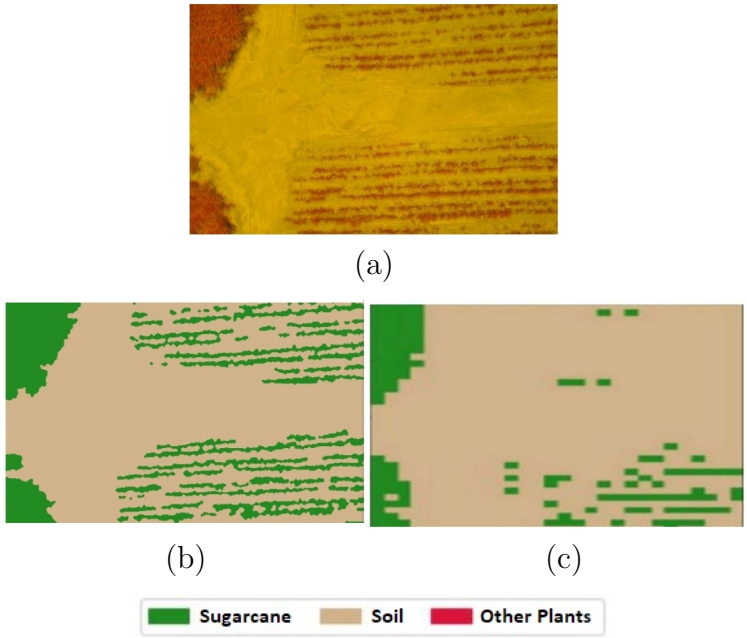


Figure 3.15: Baseline visual results. The first image (a) represents an input presented to the baseline. The left (b) and the right (c) image masks indicates, respectively, the ground truth (before being transformed) and the predicted masks.

plantation area leading to slightly different estimated cuttimes. Therefore, the prediction effect discussed is expected since the model should be capable of interpolating the cut-time values between the crops.

In Fig.3.17, which presents the results of the model trained with I_{RGB}, the classifications of sugarcane and soil were close to the ground truth, but also presented some noise. Also, it can be noticed that the other plants classification was not so precise.

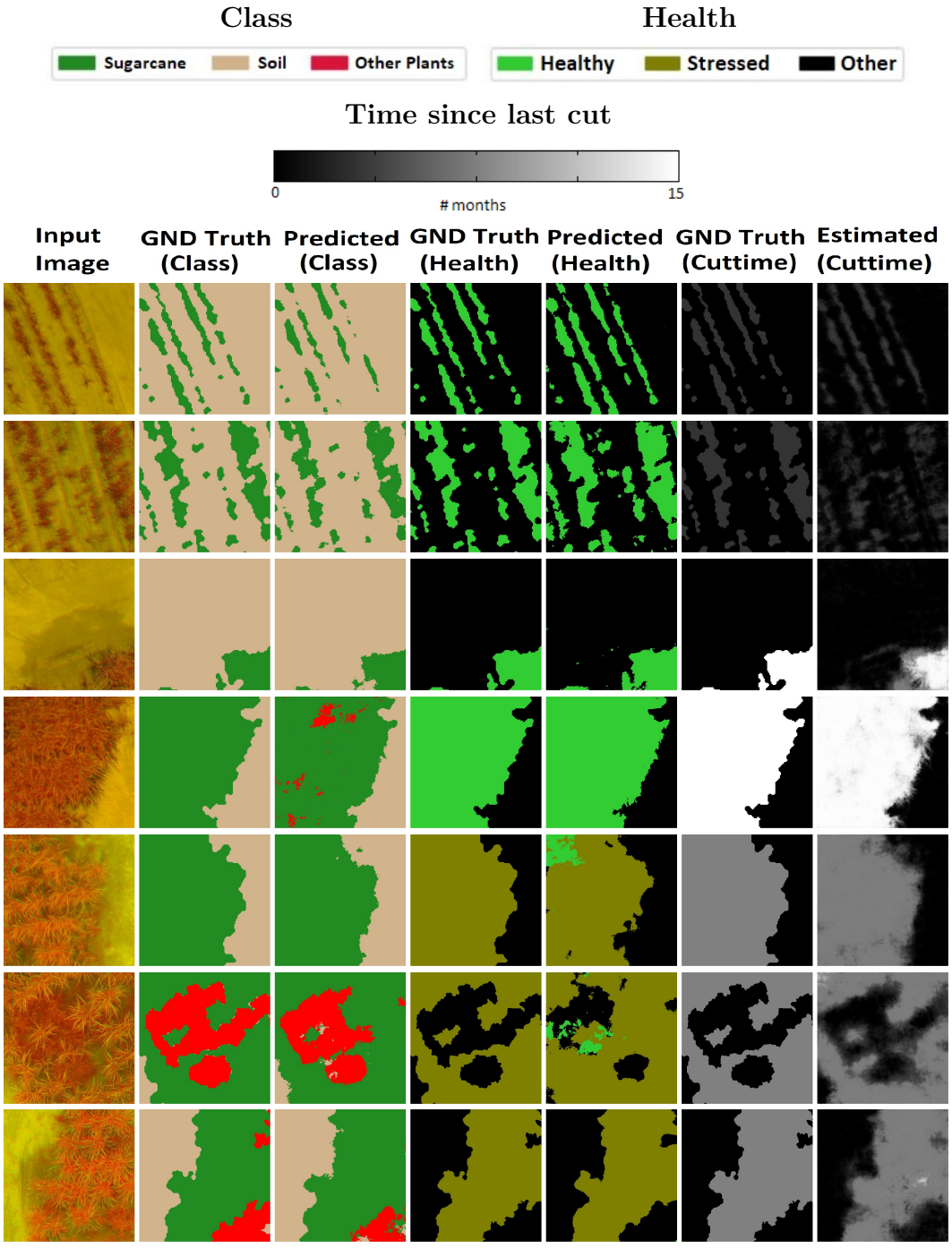


Figure 3.16: Semantic segmentation visual results for the model trained with the $I_{R(R+IR)}$ images. In the caption, the classifications for Class and Health are given by the expressed colors while the time since last cut estimate is given in a discretized grayscale.

Figure 3.17: Semantic segmentation visual results for the model trained with the I_{RGB} images. The results shown are related to the sugarcane classification only.

3.6.1 Additional Results

In order to reaffirm the previous generalization results, we captured new sugarcane, soil and other plants images from a different location at Embrapa's Farm at Rio de Janeiro of Southeast Brazil. This time, a different drone with a color camera was available, thus only the RGB model for the first classification task was tested. The visual results are shown in the figures 3.18, 3.19 and 3.20.

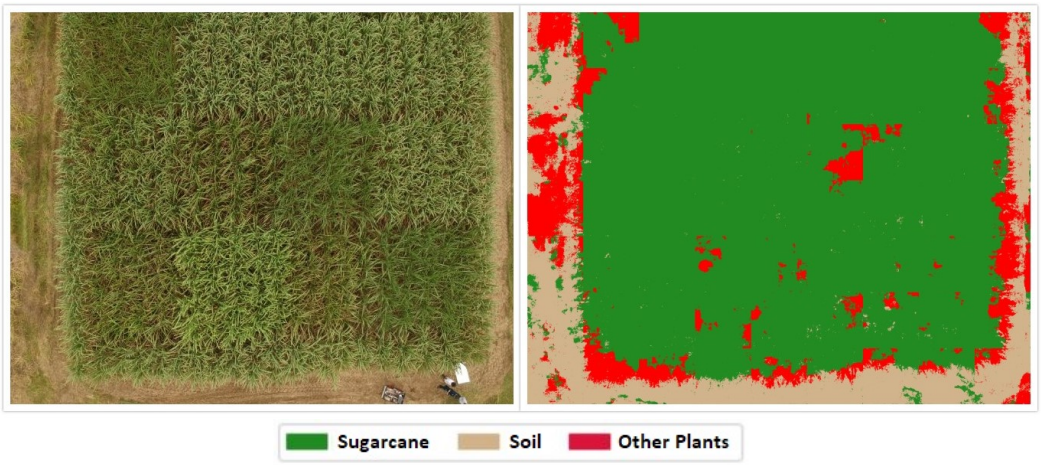


Figure 3.18: Segmentation results on different varieties of sugarcane.

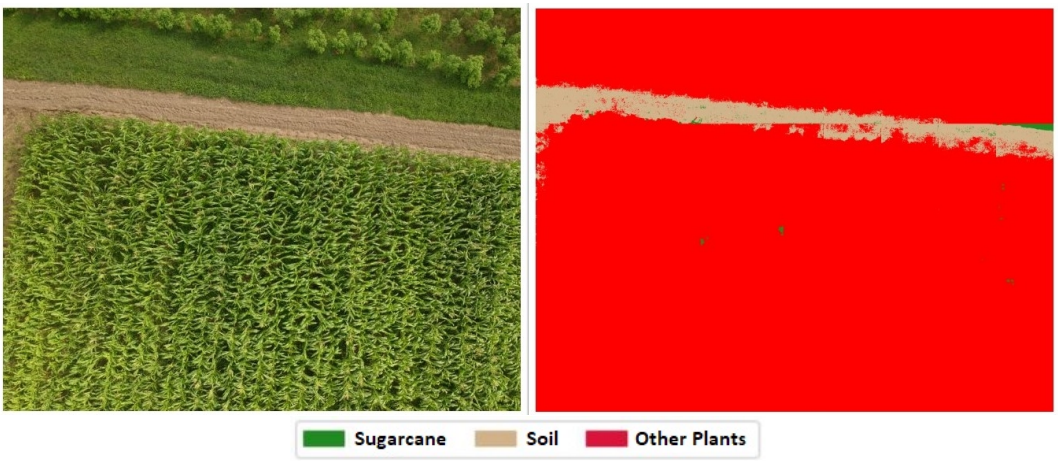


Figure 3.19: Segmentation results on cornfield.

The model was trained with a different dataset, but it could identify reasonably well a large area of "Sugarcane", "Soil" and "Other plants" for the Empraba's images samples. The cornfield and vegetable garden was correctly classified as other plants. This is an interesting result due factors, such as, the drone flew at different height compared to the original dataset; the images were not rectified; the cameras are different; the ground composition and climate are different due to the vegetation location.

Figure 3.20: Segmentation results on vegetable garden.

3.7 Conclusions and Future Work

In this paper we presented real information collected in sugarcane farms. We presented techniques to preprocess the data to improve the dataset used by the supervised models. Our vegetation monitoring solution for sugarcane farms provides relevant and precise information (multi-task segmented images) for sugarcane crop analysis. Using an UAV and an embedded computer (or remotely), the vegetation images can be captured in fractions of seconds and the crop information can be quickly obtained.

As shown in the results, due to the advantages of the visible and invisible bands for vegetation, the concatenation of the Red and the Red+IR bands was essential to provide good performance for health and cuttime. The results using the multi-task SegNet algorithm for images was relevant for vegetation segmentation tasks involving remote sensing images from UAVs.

Some ways to improve the results is to enhance the masks, although when the dataset is large enough the supervised model should be capable of dealing with this type of noise.

For future work we pretend to use the GPS coordinates to send ground robots to the location and collect data, such as, physical samples of vegetation. Also, it is intended to apply this technique to different vegetation classes and sugarcane varieties.

Acknowledgement

This work was partially funded by a Masters Scholarship supported by the National Council for Scientific and Technological Development (CNPq) at the Pontifical University Catholic of Rio de Janeiro, Brazil.

This work was partially supported by the UTFORSK Partnership Programme from The Norwegian Centre for International Cooperation in Education (SIU), project number UTF-2016-long-term/10097.

4
Automatic Visual Estimation of Tomato Cluster Maturity in Plant Rows

Keywords: Deep Learning; Object Detection; ConvNets; Regression; Image Processing; Image Analysis; Tracking.

Abstract: The present paper aims to implement image processing algorithms to accelerate and facilitate the evaluation of the harvest condition in tomato farms. In order to achieve this, two different Deep Learning models are trained and combined with counting methods to produce a harvest monitoring system for embedded applications using an Intel® Movidius Compute Stick™[1]. The first model detects the location of cherry tomato clusters, while the second estimates the fruit's maturity. The results are compared to a baseline implementation based on segmentation. Next, a multiple counting method based on region of interest is applied to the detected clusters in videos to count the tomatoes at different maturation stages. In order to produce a more robust counting, a tracking system is implemented which uses temporal information to find the unique tomato clusters in videos. In the evaluation stage, the obtained location results indicates an IoU of about 89% when using the MobileNetV1 as a feature extractor and choosing the appropriate location anchors. The maturity estimation results indicates better performance for the supervised algorithm as compared to the baseline, providing a root mean square error of $7.709 \cdot 10^{-2}$. The best results were obtained when combining the fully learned solution with the tracking system, correctly counting the majority of the tomato clusters at multiple maturation stages.

4.1
Introduction

The large demand for fruits brings challenges to farmers, such as optimizing the costs and effectiveness of manual labor for harvesting and reducing the percentage of fruit loss which contributes to the quality of the final product [61]. Hiring more seasonal manual workers or paying overtime may improve the

[1]Intel® Movidius Compute Stick™
Available: https://software.intel.com/en-us/neural-compute-stick [January 1, 2019]

production, but increases the costs or affects the workers health at the farm [62] when working for extended hours. The effectiveness of manual harvesting and the percentage of fruit loss are linked to the monitoring and control of the fruit maturation during harvest [61].

Fruit harvesting is a time-consuming task because each worker needs to observe and touch the fruits in order to evaluate their maturation and decide whether to harvest on the current or next days. Some technological methods use custom non-destructive crop monitoring sensors aiming to solve this problem by measuring the fruit ripeness [63, 64]. The detection is based on fruit reactions that take time to take effect and is prone other to dependent variables (e.g. climate change and fruit variety) that lower the robustness. Researchers have implemented a system that uses a harvesting robot that is divided into a vision system which calculates tomato cluster stalk position and an end-effector to harvest the fruit, but they do not estimate the ripeness [65, 66].

Some variety of cherry tomatoes are considered ready for harvest when the skin changes its color from greenish to reddish (i.e. red, yellow or orange) or the tomato closest to the stem comes out in hand. However, in some farms, these fruits may not be harvested individually, being necessary to harvest the entire cluster after manually estimating the ripeness that takes into account all the tomatoes on that cluster. Each cluster is packed to be sold in the supermarket even when the maturities are mixed (i.e. red tomatoes with a certain proportion of green ones).

We implemented a robust system to monitor the maturity in cherry tomato clusters on plant rows, that only requires an RGB camera and an embedded computer, being relatively cheap and easily attached to embedded systems for statistical data collection or even automatic harvest. The system is based on supervised Deep Learning (DL) techniques, being able to generalize its response to different fruits when feeding the system with new labeled data.

The dataset used in this work was collected from video snapshots captured on a tomato farm located at municipality in Vestfold county in Norway (greenhouse). The videos were taken covering three different plant rows which contained multiple tomato clusters in different light conditions, shapes, positions, different levels of maturation and quantity of tomatoes in each cluster.

Our system is divided into two supervised deep learning subsystems and two counting subsystems. The first supervised subsystem is an object detection model that learns, through regression, how to fit bounding boxes around the tomato clusters (i.e. locate the clusters) after feeding the algorithm mainly

with ground truth locations (labels manually obtained) and a set of default bounding boxes (i.e. anchors) ratios. We used a pre-trained convolutional neural network to act as a basis for the feature extractor for the object detector. This leads to some advantages and drawbacks (e.g. location precision; class accuracy; training and inference speed), depending on the network choice and the dataset. The second supervised subsystem is a regression model that learns, from manually labelled examples, how to estimate the maturation rate. This subsystem implemented uses transfer learning techniques to successfully estimate the maturation of tomato cluster data from few samples. We also implemented a variation for the second subsystem that uses an image analysis technique to segment each tomato cluster and, using the segmented pixels, mathematically estimate the maturation. Instead of dividing the maturation into discrete values and indicate whether a tomato cluster is ripe or unripe, we use continuous values to express how ripe a cluster is.

The location and the maturation estimation subsystems are combined to provide the clusters locations and maturation rates in videos for the application of two possible counting subsystems. The first counting subsystem uses a line of interest and a region of interest to count each tomato cluster found inside the region. The second counting subsystem implemented is tracking algorithm that separates the different clusters using their coordinates in the image and their maturation information in multiple frames.

Based on the information collected by the system, it is possible to quickly identify the best tomato plantation row for the harvest in a given day. Additionally, due to the possibility of fast data collection, the continual data collection can be used in the analysis for harvest prediction in the following days and to identify possible tomato growing problems.

4.2 Background

Classical image processing methods (i.e. computer vision and image analysis) for image recognition (also know as image classification), with the aid of supervised machine learning techniques, brought many impressive results in the past, such as real time face detection and pedestrian detection [67, 68]. Due to image conditions (e.g. changes in light and camera perceptive; noisy image) the image processing part affects the generalization of the supervised algorithm around the specific data. Also, when attempting to apply the same combined algorithm to different data (e.g. to detect animals), the image processing part should be modified in order to work properly, because the image features (e.g. shape, colors and texture) are very distinct. In these

applications, this occurs because the supervised algorithm used is mainly dependent on features extracted by the computer vision and image analysis methods. Another drawback is that, for the standard supervised algorithm to work accurately on images, it needs to be fed with a massive amount of labeled features and does not work well when applied to raw pixels.

Deep convolutional neural networks are supervised algorithms predominantly applied to images and have been widely used since the advances in GPU technology enabled their success at the ImageNet Large Scale Visual Recognition Challenge [23]. They have been developed for many image recognition tasks (e.g. classify, locate or even segment the images) with outstanding results. Differently from the previous methods, the CNNs are techniques that do not need heavy pre-processing specific to the dataset because they learn how to process the image and generalize to new data. Convolution is a parallel operation that uses multiple kernels (trainable filters) that learns how to extract images features. Subsequently, max-pooling simplifies the image by calculating the image max response using predefined square-cell grids. Commonly, CNNs are divided into many convolutions layers followed by operations (e.g. max-pooling) in which the first layers extract low-level features such as edges and blobs. The next layers extract mid-level features such as texture, up to high level-features (specific features to the dataset). The number of feature maps generated in each layer is proportional to the (predefined) number of kernels for that layer. After the last convolutional layer, a conventional neural network is usually attached by means of a fully connected layer. The neural network is composed of neurons, which each have trainable parameters, that learns how to combine the features maps of the last convolutions layer to predict results (e.g. classification, regression or both).

Standalone CNNs can solve the image recognition problem due to the robust advanced and parallel operations, but the machine intelligence, in this case, does not learn how to understand scenes. In other words, the machine is not able to classify and locate different objects in the same scene by itself as humans do. Researchers have developed methods based on CNNs (i.e. Regions with CNN features [69]) to improve the scene understanding issue in machines, aiming in real time applications (R-CNN [69], Fast-RCNN [70] and Faster-RCNN [71]). The first R-CNN method is divided into two different stages: the first one is the generation of image regions, using image segmentation, to find regions with high indication of being an object (i.e. region proposals) in each image. Then, for each region proposal (RP) it makes rectangular crops evolving each region and calculates features maps using convolutional neural networks. Next, using a pre-trained classifier on the same dataset, the

method classifies these features according to the predefined targets. Due to the the high number of regions generated and consecutively high number of convolutions, the algorithm was too slow for training and inference to be used in real time applications. The second method (Fast-R-CNN), is also divided into two stages, uses convolutions blocks (i.e. convolutions and their auxiliary operations) in the entire image, obtaining feature map directly. Then, it uses a Region of Interest (RoI) pooling layer that uses the RP locations to crop the feature maps into smaller features to train a two-headed neural network (i.e. a regressor to learn the object coordinates and a classifier to learn the predefined classes, both in the same network). The regressing part uses an *objectness* score method to evaluate whether the cropped feature map is an object or not, while the classifier identities the respective class of that object. This method highly improves the processing speed (by a factor of 25) due to the considerably smaller amount of convolutions applied. The third method (Faster-R-CNN) improves the results and speed (around 10 times faster than the Fast-RCNN) based on the previous methods concept, but removing the RP locations and adding an intermediate region proposal network (RPN). The RPN is pre-trained using a set of reference bounding boxes with different shapes (called *anchors*) to learn how to extract RPs from the image, making the final algorithm an end-to-end learning method. The final algorithm shares the RPN convolution layers with the previous method (Fast-RCNN) to learn the object location and class. Sharing layers optimizes the training and processing speed. The previous methods can detect objects reasonably well, but the inference time in videos is still too slow in usual GPUs (around 5 frames per second for the fastest method) for real time applications. Important algorithms in the research area were developed to solve these problems, highlighting "You Only Look Once" (YOLO) and "Single Shot MultiBox Detector" (SSD) [72, 73] that could mix the two stages into a parallel stage. The first algorithm can increase the frames per second (FPS) on inference up to 150, but compromises the accuracy of the results. The second algorithm highly improves the accuracy, but inferring at $22 - 60$ FPS which is enough for real time applications. In the SSD algorithm, starting from the first convolutional block and following to the last block, the feature maps decrease in terms of resolution, but increases in terms of depth. For each feature map extracted from the convolutions at different layers, the algorithm uses a shared regressor and a layer dedicated classifier. Consequently, the object detection is done at multiple scales being able to handle objects at different sizes [73]. In order to label a dataset for the SSD model, each object to be detected must have a class identification and its coordinates position in the image. As with the other methods, in the trainig

stage, SSD also uses anchors to learn the bounding box coordinate offsets at multiple scales, based initially on which anchors correspond to a given label object (i.e. ground truth information). The SSD architecture is explained in more detail in Sec.4.3.2.

4.3 Method

In this section, we begin by describing how the data were collected and labelled to be used in our different approaches for tomato clusters location and their maturity estimation. In sequence, we describe the model for object detection and the baseline for the estimation followed by the main regression model. Next, we discuss the implementations for counting the clusters in videos.

4.3.1 Preparing and Labeling the Dataset

The color cameras were coupled to a pipe rail trolley in order to record the videos on the farm by following a unidirectional horizontal movement, covering a different plant row in each video. After recording the videos, a certain number of snapshots were taken, as shown in Table 4.1. The snapshots are used to train supervised deep learning algorithms and measure their performance while for the final counting algorithm all intermediate frames are used as well.

Table 4.1: Snapshots taken for each video at 30 FPS

Video Name	Length	Total Number of Frames	Number of Snapshots Taken	Proportion of Snapshots Taken
Video 1	2 min 44 s	4920	118	2.398%
Video 2	1 min 28 s	2640	63	2.386%
Video 3	1 min 43 s	3090	73	2.362%

The images obtained from the snapshots are then mixed and labelled for the supervised methods for fruit detection and the maturation estimate. The major differences between the videos are the expressive changes in light conditions and the distances and angles between the camera and the fruits.

4.3.1.1 Fruit Detection

The labeling of images for object detection is commonly made with a graphical image annotation tool (e.g. LabelImg, used in this paper [2]), which

[2]Tzutalin. LabelImg. Git code (2015).
Available: https://github.com/tzutalin/labelImg [August 3, 2018]

is used to pack each object inside a rectangle (bounding box). After manually drawing a rectangle around the object of interest, the tool asks for a target for that object, which in this paper indicates a cherry tomato cluster. Each annotation made generates a new instance, and the dataset size is the number of all instances. After labeling all the clusters separately, the dataset is divided into training and validation sets as shown in Table 4.2. The remaining data is used to test the model's performance.

Table 4.2: Dataset separated into training and validation sets for object detection

Total Number of Snapshots	Number of Training Training Snapshots	Training Proportion	Number of Validation Snapshots	Validation Proportion
254	203	79.92%	26	10.24%

4.3.1.2 Fruit Maturation Estimate

For each tomato cluster labelled as an object to be detected, we give a rank score between 0 and 1. The rank is related to the level of maturation for a cluster, calculated by equation 4-1.

$$\text{rank} = \frac{N_{RT}}{N_{RT} + N_{GT}} \tag{4-1}$$

Where N_{RT} and N_{GT} are the number of, respectively, Reddish and Greenish Tomatoes in a cluster, counted manually. A continuous value for the level of maturation is useful and easily tunable, so the farmer can choose thresholds to classify the clusters into n different maturation stages (e.g. red, mixed and green tomato clusters).

After labeling all the instances extracted by cropping each tomato cluster in the snapshots, the dataset is divided into training and validation as shown in Tab. 4.3. The remaining data is separated to test the algorithm's performance.

Table 4.3: Dataset separated into training and validation sets for rank estimate

Total Number of Clusters	Number of Training Clusters	Training Proportion	Number of Validation Clusters	Validation Proportion
2550	2046	80.23%	255	10.00%

4.3.2 Supervised Algorithm for Cluster Detection

In order to locate and detect multiple tomato clusters in the same image, taking into account the processing speed and accuracy, we trained the SSD object detection algorithm [73]. The SSD allows the use of pre-trained

networks to form the convolutional base. There is a trade-off between speed and accuracy due to the different pre-trained architectures (e.g. *VGGNet* [20], *ResNet* [22], *MobileNet* [74] and *AlexNet* [23]) amount of trainable parameters and operations. We opted to use the MobileNet V1 [74] architecture as the base of the SSD because it has optimized convolution operations and good performance, therefore it can be used in real time applications. The MobileNet V1 has 28 layers formed by 13 convolution alternate blocks, being the main operations in the block the Depthwise Separable Convolutions (DSC). The basic idea of this convolution is to mitigate the computational cost generated through parallel operations by separating the heavy multiplications operations in two sums with lighter multiplications each. The DSC is divided into two different convolution layers: Depthwhise Convolution (used for filtering) and Pointwise Convolution (used to combine features). Instead of applying convolutions to whole images and feature maps channels at the same time, the depthwise convolutions layer applies smaller convolutions in each channel separately, producing features with the same number of channels as its input which are then stacked together. The pointwise convolution layer takes the pile of features and applies convolutions with kernels 1x1 (i.e. filtering pixel by pixel) for all the channels at once, creating linear combinations between the feature maps [74]. These operations are not only faster, but also significantly reduce the quantity of trainable parameters of the model.

As mentioned before, the SSD model uses a shared regressor and layer dedicated classifiers to accomplish the object detection task. In order to train the algorithm for this task, the model needs to minimize a weighted combined loss function (i.e. multibox loss) that is divided into a location loss and a confidence loss. The multibox loss function is minimized by updating the SSD trainable parameters using backpropagation [75]. The first loss function measures how close the predicted bounding box coordinates (based on the set of anchors) are from the ground truth. The second loss function computes how confident the model is in relation to the class predicted, given the ground truth, in a predicted bounding box [73]. In this paper we used the detector to find a single class, therefore the confident loss function is simplified. During the training phase, the SSD uses initially a matching strategy that finds the best set of anchors to be used for each ground truth to train the network [73]. The regression part of the model predicts the bounding box offset coordinates ($\Delta(cx, cy, w, h)$) based on the best anchors found. For multiple bounding boxes in the same image, the anchors are handled based on a matching approach [76].

The pre-trained network to be used as a base for the feature extraction is usually split in two parts and the layers depth of each part is chosen manually.

After obtaining the features maps generated by the first part, a layer that contains the regressor and a convolutional classifier is applied. The same logic is applied to the second part and consecutively to auxiliary convolutional layers as shown in Fig.4.1. As a consequence, multiple bounding boxes pointing to the same object, at different scales, are generated for each predicted class. The results are combined using non-maximum suppression to remove the detections with location and detection scores lower than predefined thresholds.

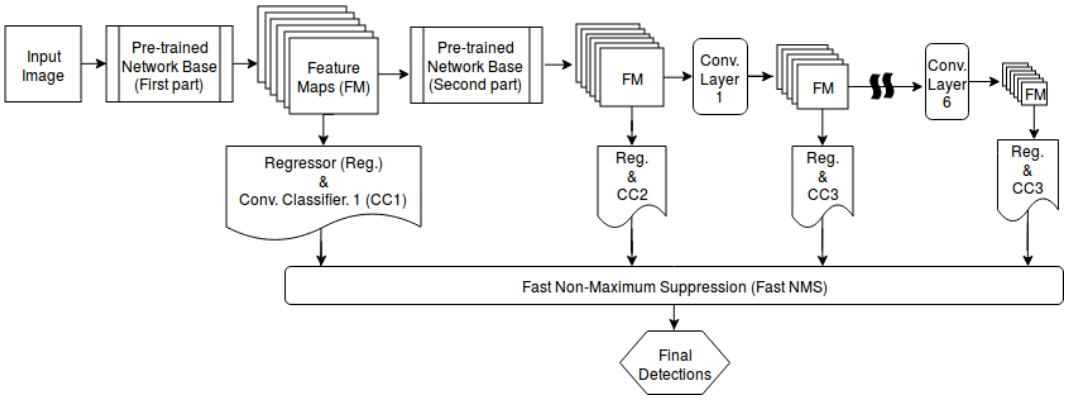


Figure 4.1: SSD architecture for generic pre-trained network used as base (feature extractor). The schematic illustrates the pre-trained network division in two parts, where each part is used to extract features that are passed to the first two respective regressors and classifiers. The next feature extractors are auxiliary convolutional layers in which features produced are passed to more respective regressors and classifiers. The multiple locations and classifications for the same object are combined to calculate the final detections.

4.3.3 Cluster Evaluation

After obtaining the bounding boxes for each cluster of tomatoes, it is intended to identify how ripe the clusters are. In order to complete this task, two different approaches were implemented and tested. The first approach consists in implementing a segmentation algorithm, based on the Ohta and Otsu algorithms [46, 77] to calculate the level of maturation using the segmented pixel areas. The second approach is related to the use of CNNs adapted to provide a regression output which role is to estimate the level of maturation. In both cases, the methods are applied to each cluster sub-image inside the bounding boxes.

4.3.3.1 Image Analysis Method

The main idea to use image analysis algorithms is to separate by segmentation the reddish and greenish tomatoes in a real environment, taking

advantage of the theirs colors and filtering the agricultural background. The major advantage of these algorithms is the possibility of its direct application without the need for labeling, training the data and availability of GPUs.

An usual image segmentation technique used in image analysis is called color segmentation, which allows the separation of an image into different sets of pixels delimited by thresholds chosen from the image histogram. It could be used to separate objects in the image such as the reddish and greenish tomatoes. However the same threshold to segment a similar object varies depending on factors such as background noise, light conditions and variations in object color. A practical and efficient solution is to use adaptive threshold algorithms (e.g. Otsu [46]) for image segmentation. Another considerable problem concerns the complex background of a tomato farm, with color intensities (green leaves colors) close to the color of the green tomatoes. A method that segments red and green fruits under complex agricultural background combines the previous technique and also changes the color space from RGB to that of Ohta and Wei, which is more robust to light change and mitigates the background disturbance [46, 77, 78]. As discussed by the authors, the best Ohta and Wei features for fruit extraction are I_2' and I_W [78]. The equations 4-2 and 4-3 represents the features I_W and I_2' that combine the Red (R), Green (G) and Blue (B) channels of the RGB color space. The equations are applied to the tomato clusters sub-images inside the bounding boxes.

$$I_W = R - G \tag{4-2}$$

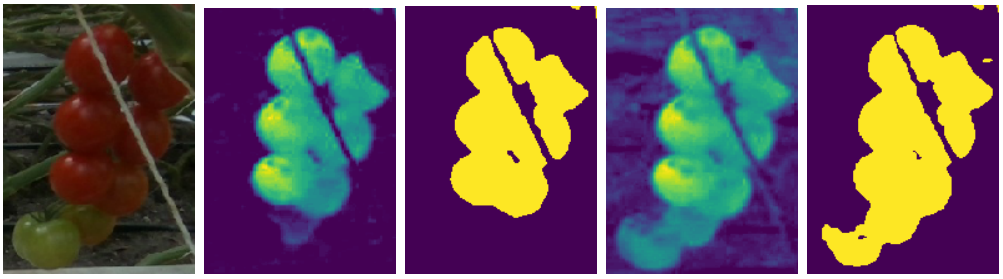
Subtracting the R and G channels results in an image with only reddish tomatoes and low background noise.

$$I_2' = R - B \tag{4-3}$$

Subtracting the R and B channels maintains the reddish and greenish tomatoes and removes part of the background. For this feature, the contrast level between the green tomatoes and the background is highly noticeable.

After changing the color space, the adaptive threshold is applied to the features I_W and I_2'. It is relevant to notice that the adaptive thresholds have improved results when applied to smaller areas, as the ones cropped using the bounding boxes coordinates, because the intra-class variance, used to find the best thresholds, becomes a well behaved function facilitating the search of the optimal thresholds for the color segmentation. In order to remove noise and filter the remains of the background, we applied a Gaussian filter and opening morphology algorithm to each feature. At the end, the first feature I_W segmented and filtered provides a mask of the reddish tomatoes, while the feature I_2' segmented provides a mask of the whole tomato cluster. Subtracting

both results in a mask of the greenish tomatoes as shown in Fig.4.2 example.



4.2(a): I_m 4.2(b): I_W 4.2(c): $I_{W\,mask}$ 4.2(d): I'_2 4.2(e): I'_{2mask}

4.2(f): $I'_{2mask} - I_{W\,mask}$

Figure 4.2: Intermediate images generated applying the mathematical equations (Eq. 4-2 - 4-3) and the auxiliary operations. The first image (a) is the original image used in this example. The second (b) and fourth (d) images are the I_W and I'_2 outputs. The third (c) and fifth (e) images are the masks of each output after applying the segmentation algorithm. The last image (f) shows the masks subtracted.

After obtaining the masks for the reddish and greenish tomatoes, the maturity level (rank) can be calculated by equation 4-4.

$$\text{rank} = \frac{N_{RTP}}{N_{RTP} + N_{GTP}} \tag{4-4}$$

where N_{RTP} and N_{GTP} are the number of, respectively, Reddish and Greenish Tomato Pixels.

4.3.3.2 Supervised Deep Learning Method

In order to estimate the maturation levels more robustly, we use CNNs that, in simple words, learns how to filter efficiently the images contained inside the bounding boxes extracting features that it considers important. Instead of training the CNN from scratch, we use the VGG-16 and MobileNet pre-trained networks [20, 74]. The VGG-16 and MobileNet architecture differs mainly in terms of convolutional operations, number of convolutional layers and number of trainable parameters which results in a trade-off between performance and training/inference speed. The VGG-16 usually provides better generalization

compared to MobileNet, but takes much more time training and processing images [79]. Also, a transfer learning [19] technique is used, which allows the transferring of pre-trained parameters learned on the first *"Y"* layers. Next, we use a fine tuning method that permits the freezing of layers, marking the *"Y"* layers as non-trainable (frozen) and the next layers are marked as trainable during model learning. The advantages of using transfer learning are the faster training time and the significant lower number of trainable parameters which also prevents overfiting.

The original pre-trained networks were trained using the ImageNet dataset for the *Large Scale Visual Recognition Challenge* [24] which contains a thousand classes used for binary classification. The pre-trained output layer contains a number of neurons equal to the number of classes and it uses the softmax activation function applied to the layer, therefore the classification of a given input image is made choosing the neuron with higher activation. As mentioned before, for maturation estimate, we are looking for a continuous value instead of a binary value. Thus, we modified the output layer for regression, keeping a single neuron and changing the activation function to sigmoid. Consequently, the output value of this neuron varies between 0 and 1 according to the maturation rate range. The Fig.4.3 shows the pre-trained models adapted for regression.

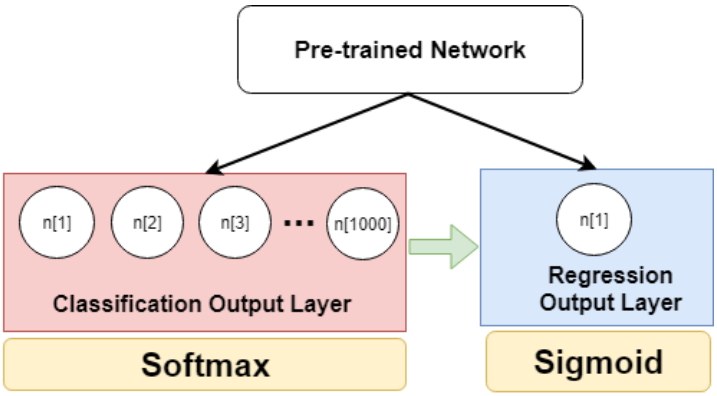


Figure 4.3: Pre-trained architecture adapted as a regression model. The pre-trained network block represents the original convolutional layers while the following layers are modified. The red box indicates the original neurons for a classification task and the blue box indicates a single neuron that is used for the regression.

Other modifications that needs to be done to convert the classification model into a regression model are the loss function and the metric. The loss function is optimized during training to decrease the errors between the maturation values predicted by the model and the ground truth values. The

metric is used to measure the model's performance. We use the Mean Square Error (MSE) regression loss function as shown in equation 4-5.

$$MSE = \frac{1}{bz}\Sigma_{i=1}^{bz} e^2\,, \quad e \in \mathbb{R}^n\,, \quad bz \in \mathbb{Z}_{>0}\,, \tag{4-5}$$

where e is the training error and bz is the number of samples used per training step (batch size). The metric we use for evaluation in this regression task is the Root Mean Square Error (RMSE). While the MSE is applied to the training set, the RMSE is applied to the validation and test sets.

4.3.4 Tracking Clusters in Videos

In order to track individually each tomato cluster in videos, we have implemented a logic algorithm that takes advantage of each cluster centroid coordinates (x, y) related to the image and its rank computed in the last N frames of video. Our tracking algorithm, based on multiple frames, brings advantages, such as occlusion robustness and multiple counting avoidance for the same cluster.

We use a Degree of Compatibility (DoC) computed using a weighted Euclidean Distance (wED), as shown is Eq.4-6, to measure how closely matched the clusters in the current frame (IC) are to the all the clusters being tracked in the last N frames (CC). The IC with the lowest DoC value (BC) and below are added to the CC list and receive an identification (id). Also, the number of times ($nTimes$) that each IC was found in CC is counted and the referent CC centroid and rank are updated. The rank update is based on a smoothed equation which gives more emphasis to the rank of the newest clusters found in the list, but also considers the oldest ranks as shown in Eq.4-7. In order to save all the different clusters tracked in the video, the tracking system saves each unique cluster of the CC list when each is matched over $N/2$ times. The algorithm 1 shows a pseudo code for the tracking system.

$$\begin{aligned}\mathrm{DoC}_{ij} &= \mathrm{wED}([CC_i.\mathrm{x}, CC_i.\mathrm{y}, CC_i.\mathrm{rank}] \cdot a_w, [IC_j.\mathrm{x}, IC_j.\mathrm{y}, IC_j.\mathrm{rank}] \cdot a_w) \\ &= \sqrt{[(CC.x - IC.x) \cdot a_x]^2 + [(CC.y - IC.y) \cdot a_y]^2 + [(CC.r - IC.r) \cdot a_r]^2}\,,\end{aligned} \tag{4-6}$$

where a_x, a_y and a_r are weighted values (between 0 and 1) that are manually chosen to give more emphasis to the clusters chosen elements and $\vec{a}_w = [a_x, a_y, a_r]$. The centroids and ranks elements are normalized to provide values between 0 and 1.

$$\mathrm{SmoothRank} = \frac{CC_i.\mathrm{rank} \cdot (CC_{\mathrm{id}}.\mathrm{nTimes} - 1) + BC_{similar}.\mathrm{rank}}{CC_i.\mathrm{nTimes}} \tag{4-7}$$

4.3.5 Counting Clusters in Videos

In order to count quantitatively the number of tomatoes clusters in videos at different maturation stages, we established thresholds for the maturation levels to divide it into the following stages: Red, Mixed and Green Clusters. The Red and Mixed Clusters are considered ripe while the Green Cluster is not ready for harvest. Table 4.4 shows the thresholds chosen in the division.

We implemented two different methods to count the tomato clusters. The first method counts the clusters using a Region of Interest (ROI) while the second method uses the tracking system. Both methods uses the cluster detection algorithm to locate and extract each cluster from the image and the cluster evaluation algorithms to estimate the ripeness.

4.3.5.1 Method 1: Using a Region of Interest

As mentioned before, the clusters image sequence from the videos are constantly moving to the left which facilitates the application of methods based on region of interest to count the objects at each frame. The first method uses an object counting API, proposed by [80], in which the cumulative counting mode uses a vertical line of interest to count one or more objects whenever it pass between a region around the line (i.e. Region Of Interest (ROI)). Mathematically, the method considers an object to be counted when the center line of the bounding box is aligned with the line of interest around an arbitrary deviation (ΔR) as shown in Fig. 4.4.

4.3.5.2 Method 2: Using the Tracking System

The second method takes advantage of the tracking system using the clusters *id* information to count the number of unique tomato clusters at the mentioned maturation stages.

4.3.5.3 Ratio of Ripe Clusters

In order to evaluate the maturation of an entire row (using the three maturation stages), we calculate a ratio (rT) as expressed in equation 4-8.

$$rT = \frac{R + M}{R + M + G}, \quad \{R, G, B\} \in \mathbb{Z}, \tag{4-8}$$

where R, M and G are the number of, respectively, Red, Mixed and Green tomatoes clusters.

Algorithm 1: Tracking algorithm implemented

```
1  function Cluster Tracking (IC, CC);
   Inputs : ImageClusters (IC- Clusters in the Current Image);
            CurrentClusters (CC- Clusters tracked in the Current N frames)
   Output: Clusters Tracked (CT- All the clusters tracked)
2  for all CC do
3      BestDoC = −∞
4      BestCluster = [ ]
5      for all IC do
6          DoC = wED([CC.x_i, CC.y_i, CC.rank_i] · a_w, [IC.x_j, IC.y_j, IC.rank_j] · a_w )
7          if DoC < BestDoC then
8              BestDoC = DoC
9              BestCluster = CC_i
10     if BestDoC < threshold_1 then
11         BestCluster.id = CC_i.id
12         Increment the nTimes (number of times) the CC_i was found
13         CC_i.centroid ← BestCluster.centroid
14         CC_i.rank ← SmoothRank(BestCluster.rank, CC_i.nTimes)
15         Remove BestCluster from the IC list
16 for all IC do
17     if IC_j appears in the three quarter of the image (3/4 · width) then
18         for all CC do
19             DoC = wED([CC.x_i, CC.y_i, CC.rank_i] · a_w, [IC.x_j, IC.y_j, IC.rank_j] · a_w )
20             if DoC > threshold_2 then
21                 IC_j gets a new id
22                 add IC_j to the CC list
23 for all CC do
24     if CC_i was matched N frames ago then
25         Remove the CC_i from the CC list
26         if CC_i.nTimes > N/2 then
27             add CC_i to the CT list
28 return CT;
```

Table 4.4: Range of values to separate the maturation rates into three maturation stages.

Condition	Cluster Classification
rank $\geq 70\%$	Red Cluster
$30\% \leq$ rank $< 70\%$	Mixed Cluster
rank $< 30\%$	Green Cluster

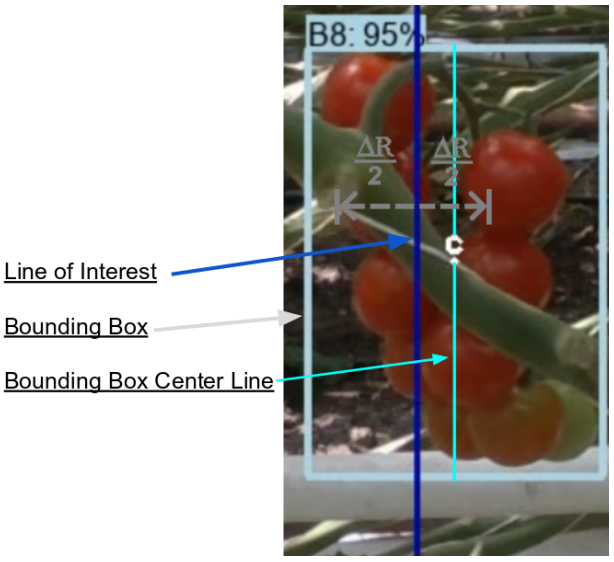


Figure 4.4: Example of the TensorFlow Object Counting API in Cumulative Counting Mode. The line of interest is fixed in the vertical and horizontal positions while the bounding box center line moves according to the objects centroid.

4.4 Experiments

The supervised experiments performed in this work were carried out on NVIDIA® Geforce P100™ GPU, that made possible the parallelization of CNN's operations, considerably increasing the speed at which the training was done. For the supervised algorithms at inference stage on the test set and on the videos we used Intel® Core™ i7 CPU and Intel® Movidius Compute Stick™.

The experiments are divided into location, level of maturation estimate and clusters counting. The two first kind of experiments were done separately while the last kind was combinedly done.

4.4.1 Object Detection Experiments

The detection experiments were based on the TensorFlow API for object detection which made possible to handle the detection behavior in different experienced configurations. We analyzed the impacts of two CNNs and its features extractors (CNN layers) used as a base of the location algorithm. Also, we tested different values for the training hyperparameters which led to better detection results. The default bounding box generator (anchor generator) hyperparameters were chosen properly to allow the algorithm to learn to detect tomato clusters at different shapes and in partial occlusion. We prioritize the choice of anchors with higher relevant vertical ratio (≥ 1), due

to the most common cluster's shape (height greater than width). However, for partial occlusion cases the algorithm should be able to detect most part of the clusters, so also we also chose lower vertical ratio values (≤ 0.5). The two best configurations are presented on Table 4.5.

Table 4.5: Best two configurations for the detection model.

Configuration	**Experiment**	
	E1	**E2**
Base CNN	VGG16	MobileNet V1
Base CNN Layers (First Part)	$1-4^{th}$ Conv.	$1-6^{th}$ DSC
Base CNN Layers (Second Part)	$4-5^{th}$ Conv.	–
Anchor generator aspect ratios	0.33; 0.5; 1.0; 2.0; 3.0	0.33; 0.5; 1.0; 2.0; 3.0
Batch size	1	1
Initial LR (η)	10^{-5}	10^{-4}
Epochs	120	100

In order to reduce the oscillation of errors during training given the high quantity of tomato clusters per image (around 10) we chose lower values for the batch size. The same effect occurs for the learning rate, being necessary for the algorithm to converge by taking smaller steps. As a consequence, a greater number of training epochs is required.

Additionally, we applied data augmentation algorithms to the training set to avoid overfitting at the training phase by increasing the amount of trainable data, which improves the network's ability to locate objects in the test set. The two transformations used were horizontal flip and random crop. The second transformation was also helpful to detect smaller tomato clusters in images.

4.4.2 Level of Maturation Experiments

The image analysis experiments followed a tuning of the Gaussian filter hyperparameters and the structure element kernels sizes for the opening morphology, aiming to obtain suitable segmentation for the I_W and I'_2 features.

For the supervised method, the VGG-16 [20] and MobileNet [74] networks were used in our experiments by transferring their pre-trained parameters and keeping some layers frozen (non-trainable) to train the regression model. Similar to [1], we verified and chose the suitable number of layers to be frozen ($''Y''$) for each pre-trained model. All the supervised experiments were made using the sigmoid function for the output neuron that is attached to the last regression layer.

4.4.3 Counting Experiments

After obtaining the object detection and level of maturation best configurations, we combine these algorithms to be used by the counting methods in videos. Differently from the previous experiments where the test data was used, the counting experiments uses the whole three videos as samples. The data depends on the location algorithm to find the tomato clusters so the maturation can be estimated. The ground truth in this case is obtained by counting manually the number of detected tomatoes in each video at the three maturation stages. Thus, the undetected clusters are discarded from the experiments.

The number of counting experiments for each video is given by the combination of the detection algorithm, the level of maturation algorithms and counting method as shown in Table 4.6.

Table 4.6: Experiments for each video.

Object Detection	Level of Maturation	Counting Method Base	# Experiment
Deep Learning (DL)	Image Analysis (IA)	ROI (M1)	DIR: (DL+IA+M1)
		Tracking (M2)	DIT: (DL+IA+M2)
	Deep Learning (DL)	ROI (M1)	DDR: (DL+DL+M1)
		Tracking (M2)	DDT: (DL+DL+M2)

Also, for the counting methods, we made pre-experiments varying some of their hyperparameters. The tracking DoC hyperparameters (a_x, a_y, a_R) were tuned sufficiently so the tracking system provides satisfactory visual results in the videos. The most representative cluster information evaluated when using the fully supervised algorithm was the maturation rate while for the image analysis method was the vertical component of the centroid. This is due to the higher robustness of the regression model to evaluate the clusters maturation. We verified that the deviation values (ΔR) for the ROI method must not be low because the algorithm would miss an expressive amount of bounding boxes due to the pixel shift between frames that may be larger than the deviation. The opposite effect occurs when using a larger value for the deviation, leading to multiple counting for the same object. The Table 4.7 shows the range of values for the base counting pre-experiments.

Table 4.7: Range of values for the hyperparameters in the pre-experiments.

Counting Method Base	Hyperparameter	Range of Values		
		Min	Max	**Optimal***
Tracking (for DL)	a_x	0	0.6	0.2
	a_y	0	0.6	0.5
	a_R	0	1	0.6
Tracking (for IA)	a_x	0	0.8	0.3
	a_y	0	1	0.5
	a_R	0	1	0.3
ROI	ΔR (pixels)	1	50	0.6

*The optimal value is the best estimate found for the hyperparameters

4.4.4 Movidius Experiments

For the Movidius experiments, we evaluate the inference speed for the regression, the object detection and the combined models. In order to achieve this in the Intel® Movidius Compute Stick™, we initially convert the Tensorflow models to the Caffe framework using a converter implementation[3] and exporting the trained parameters from the Tensorflow model to the converted model. The Movidius sdk version used (ncsdk v2) allows the upload of multiple graphs to a single neural stick by allocating the model's graphs with FIFOs, being feasible to group the combined model.

4.5 Results and Discussions

In the results section, we present the partial results for cluster location and level of maturation estimate. Then, for the combined approaches, we show the complete results. The processing time for embedded applications and the additional results are presented and discussed.

4.5.1 Object Detection Results

In order to evaluate the object detection algorithm performance on the test set, we used the Intersection over Union (IoU) metric. The metric to measure the class performance is not necessary because we are only interested in locating the tomato cluster class. The IoU calculates how close a predicted bounding box is to the ground truth, using Eq.4-9. The method is illustrated in Fig. 4.5. When the boxes are closely matched, the IoU assumes values close to 1, but when they are closely unmatched it assumes values close to 0. Also,

[3]tensorflow2caffe
Available: https://github.com/lFatality/tensorflow2caffe [January 22, 2019]

whether the model predicts part of the image that is not correctly a predetermined class, the IoU assumes value 0.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}, \qquad IoU \in \mathbb{R} : 0 \leq IoU \leq 1, \tag{4-9}$$

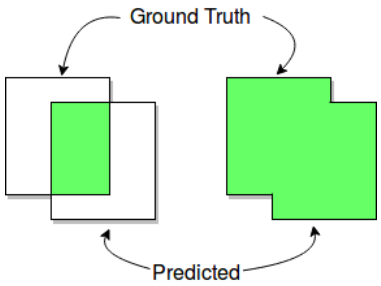


Figure 4.5: Intersection over Union (IoU) for bounding boxes. The green color on the left image represents the area of overlap, while on the right image it represents the area of union.

The final IoU value for the test set is given by the mean of all the IoU as shown in Eq. 4-10.

$$IoU_{final} = \frac{\sum_{i=1}^{N_{pred}} IoU_i}{N_{pred}}, \qquad IoU_{final} \in \mathbb{R} : 0 \leq IoU_{final} \leq 1, \tag{4-10}$$

where N_{pred} is the number of predicted objects.

We also checked the sensitivity of the object predictions for the test set (excluding the non-object predictions because this penalty is already calculated by the IoU). The location results for each experiment is shown on Table 4.8.

Table 4.8: Location results for the test set which contains 175 samples.

Experiment	Correct Matches	IoU
VGG-16	157 (89.71%)	0.776
MobileNetV1 CNN	168 (96.00%)	0.892

The results show that the performance for the location algorithm using the initial layers of the MobileNetV1 CNN as a feature extractor was better than the VGG-16. The VGG-16 has much more trainable parameters, more effective convolutions and usually provides better results, but in this application MobileNet has excellent results even with its faster convolutions. As mentioned, the object detection algorithm was only used to locate a single class (cluster or non-cluster) which seems to have helped MobileNet to extract features.

4.5.2 Level of Maturation Results

Firstly, we evaluate some examples of the image analysis visual results for common and extreme cases of the segmentation algorithm performance. The figures 4.6, 4.7 and 4.8 display relevant results to be analyzed.

(a) (b) (c)

Figure 4.6: Segmentation visual results: (a) Original Image; (b) Red and (c) Green Tomatoes Segmentation with Overlay.

It can be observed that for all the examples, the red tomatoes were correctly segmented by the algorithm with very little amount of false positive pixels and false negative only in the case of specular reflection (Fig 4.7(e)). The segmentation for the greenish tomatoes in the figures 4.6 and 4.7 performed less well due to the complex background. However, the green tomatoes segmentation on the darker images (Fig. 4.8) was not satisfactory.

In order to evaluate quantitatively the supervised and image analysis methods we use the Root Mean Square Error (RMSE) metric to verify the maturation errors on the test set. The RMSE is calculated by

$$\text{RMSE} = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(\text{rank}_{\text{est}} - \text{rank}_{\text{gt}})^2}\,, \quad \text{RMSE} \in \mathbb{R}_{\geq 0}\,, \tag{4-11}$$

where rank_{est} and rank_{gt} are the estimated and ground truth maturation rates. The metric is applied to all the n clusters on test set. The comparative RMSE results are shown in Table 4.9

Table 4.9: Level of maturation comparative results.

Method	RMSE
Image Analysis	$2.153 \cdot 10^{-1}$
Supervised (DL)	
MobileNetV1	$2.522 \cdot 10^{-1}$
VGG16	$\mathbf{7.709 \cdot 10^{-2}}$

The table shows that the image analysis performance for maturation estimation was slightly better than the deep learning result involving the Mobilenet pre-trained CNN. This is possibly linked to two factors: firstly as discussed earlier, the image analysis method can segment very well the red tomatoes and reasonably well the green tomatoes. Secondly the use of the faster CNN in exchange for accuracy may not have been convenient to solve this task efficiently. On the other hand, the RMSE value for the VGG16 pre-trained results was about 3 times smaller than the image analysis method, showing that this supervised model is much more robust in adverse image conditions.

We also compare the maturation estimate (rank) distribution for the test set between the image analysis method and the VGG16 model, providing a scatter plot graph (Fig. 4.9).

The blue straight line indicates the rank linear relationship between the two methods. The vertical green lines represent their standard deviations. It can be noticed that the rank values for the supervised algorithm are more distributed while the values for image analysis are more concentrated to rank values above 0.8 ($\text{rank}_{\text{IA}} > 0.8$). Also, for low rank values ($\text{rank}_{\text{sup}} < 0.4$) the standard deviation is high because the image analysis does not segment precisely all the green tomatoes and consequently, in most cases, the estimate tends to higher values. The results are similar for both methods when the rank is between 0.7 and 0.9 ($0.7 < rank_{sup} < 0.9$) with low deviation.

4.5.3 Counting Results

In order to evaluate the experimental results, we first discuss the individual tomato clusters counting at the three denominated stages for each video presented in Tables 4.10 to 4.13. Then, we augment the discretization of the maturation stage from 3 to 10 and analyze the histogram results for each video using the best counting method.

Table 4.10: DIR results (DL+IA+M1).

Video	Ground Truth				Predicted			
	Red	Mixed	Green	rT	Red	Mixed	Green	rT
1	98	48	18	89%	100 (+2)	5 (-43)	7 (-11)	90% (+1%)
2	51	17	10	87%	41 (-10)	2 (-15)	5 (-5)	90% (+3%)
3	50	25	11	87%	41 (-9)	7 (-18)	3 (-8)	87% (+0%)

The DIR results shows that the rT ratios for all the videos were very close to the ground truth (defined in Sec.4.4.3), but the individual clusters counting errors were very high. In the worst case for video 1, even with low

Table 4.11: DIT results (DL+IA+M2).

Video	Ground Truth				Predicted			
	Red	Mixed	Green	rT	Red	Mixed	Green	rT
1	98	48	18	89%	137 (+39)	8 (-40)	9 (-9)	94% (+5%)
2	51	17	10	87%	61 (+10)	4 (-13)	5 (-5)	93% (+6%)
3	50	25	11	87%	69 (+19)	10 (-15)	3 (-8)	96% (+9%)

Table 4.12: DDR results (DL+DL+M1).

Video	Ground Truth				Predicted			
	Red	Mixed	Green	rT	Red	Mixed	Green	rT
1	98	48	18	89%	66 (-32)	31 (-17)	19 (+1)	84% (-5%)
2	51	17	10	87%	33 (-18)	8 (-9)	5 (-5)	89% (+2%)
3	50	25	11	87%	30 (-20)	13 (-12)	10 (-1)	81% (-6%)

Table 4.13: DDT results (DL+DL+M2).

Video	Ground Truth				Predicted			
	Red	Mixed	Green	rT	Red	Mixed	Green	rT
1	98	48	18	89%	95 (-3)	44 (-4)	15 (-3)	90% (+1%)
2	51	17	10	87%	48 (-3)	16 (-1)	7 (-3)	90% (+3%)
3	50	25	11	87%	50 (+0)	24 (-1)	9 (-2)	89% (+2%)

rT error, the algorithm missed 43 mixed and 11 green clusters. The combined algorithm could only provide good counting results for the red clusters in video 1 and green clusters in video 2. After changing the counting method from ROI to Tracking (DIT), the individual counting results got worse resulting in a worse rT. On the other hand, the DIT sum of all individual clusters is much closer to the ground truth for all the videos than DIR. It indicates that the tracking algorithms loses less tomatoes in general, but the segmentation was not enough to separate the maturation stages correctly.

In general, the DDR counting results seems to be better than the DIT and worse than DIR. However, the DDR results are more distributed along the maturation stages providing better counting for the mixed and green clusters.

When comparing the last counting results (DDT) to DIR, it can be noticed that the DDT shows similar rT values for the videos 1 and 2, but a slightly loss on precision for the ratio in video 3. However the individual counting was very precise which provides a much more reliable rT and, consequently superior results than all the other experiments. The Table 4.13 also shows that DDT had null counting error in the best case (on video 3) and four counting errors in the worst case (on video 1, the longest video).

The following histograms (Fig. 4.10, 4.11 and 4.12) represent the maturation estimate results that were discretized in one decimal for the two best counting experiments (DIT and DDT configurations) in each video. Expanding the range of values could help to identify in more details the counting errors

Video	Average Continuous Maturity	
	DIT	DDT
1	84.12%	68.39%
2	84.58%	71.49%
3	83.77%	66.43%

Table 4.14: Average Continuous Maturity for the DIT and DDT configurations for each video.

previously presented when using the image analysis method versus when using the deep learning model. The greenish area of the histogram is related to the green clusters, the yellowish is related to the mixed clusters and the reddish is related to the red clusters. The sum of all the clusters inside an area is similar to the values presented on the tables of the experiments which use the tracking algorithm (DIT and DDT).

In the three histograms, the experiments using the image analysis methods show that there is a tendency in choosing higher values for the rank ($rank \geq 0.9$) instead of more values in the range $0.6 \leq rank \leq 0.8$ (as the deep learning results do) due to the reasonable segmentation for the greenish, but great segmentation results for the reddish tomatoes.

For videos 1 and 2, the image analysis used as a base of the counting method, could provide better low ranking results ($rank < 0.3$). So we analyzed the two videos and noticed that the presence of lighter greenish clusters is greater than the presence in video 3 in which the clusters have a higher contrast with regard to the agricultural background and, consequently improves the segmentation results.

For the videos 1 and 3, in general, the counting results are well distributed for both methods, showing a greater number of correspondences. However, the correspondences proportion between the methods for each discretized rank is still low due to the higher counting loss in DIT when the tracking tries to find the unique clusters with the less precise rank estimate.

In order to decide precisely which plant row the harvest should start and which should be skipped, we can calculate and verify the average of the row continuous maturity (instead of discrete values) for the two best counting configurations (DDT and DIT) as shown in table 4.14.

Based on the results using the DIT configuration, the decision is not clearly highlighted because this configuration leads to high ranking values which affects negatively the average continous maturity. On the other hand, when analysing the best configuration results (DDT) we can infer that the harvest should follow the row sequence 2, 1 to 3.

4.5.4 Processing Time Results

To verify the possibility of employing the model in embedded applications, we evaluated the inference time when using a desktop which CPU compared to the Neural Compute Stick. The processing results are shown in Table 4.15.

Table 4.15: Average inference time results. The frame per second (FPS) shown is the frame processing average of all the three videos. For the regression model inference, each image represents a tomato cluster.

Model	**Average Inference Ratios**	
	Desktop CPU	**Movidius**
Regression	18 images/s	6 images/s
Object Detection	15.29 FPS	4.53 FPS
Combined	3.47 FPS	1.12 FPS

It can be noticed that the performance for the neural stick is reasonable for the object detection, but a bit slow for the combined model. This effect is a consequence of the regression model which has an expressive number of parameters that limits the processing FPS even for the Desktop CPU. To solve this problem, a parallelization of neural compute sticks could be made which should increase the inference speed or another embedded solution could be used (e.g. Movidius Compute Stick 2™, NVIDIA® Jetson™[4]).

4.5.5 Additional Results

We also verified some tracking results for occlusion cases in the videos and their impacts on maturation rate when using the DDT configuration. The results are shown in the figures 4.13-4.16.

It can be noticed that the object detection algorithm on Fig. 4.13 predicted the bounding boxes even with occlusion, but the maturation rank was affected when the occlusion was occurring which decreased the rank value for a while. On the other hand, our tracking algorithm uses the rank information between 20 frames and calculates a smoothed maturation function which mitigates this rank discrepancy.

Besides the same effect happens on Fig. 4.14, the rank estimate was higher in the first image because of the cluster behind, but the rank discrepancy is also mitgated by the tracking algorithm.

[4]NVIDIA® Jetson™
Available: https://www.nvidia.com.br/object/jetson-tk1-embedded-dev-kit-br.html [March 29, 2019]

The Fig. 4.15 results shows a moment when the object detection algorithm loses the detection due to occlusion, then detects the cluster again. The tracking system could identify correctly the two detections as the same cluster. An important point is that in the uncropped image, five other clusters were being tracked.

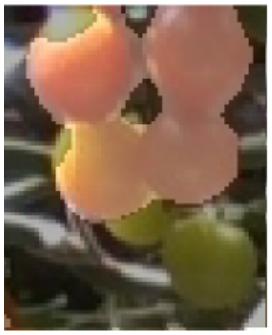
The Fig. 4.16 results shows the same cluster present in 4.15 passing through a second occlusion, losing the detection and having the rank affected by the green cluster. However, the tracking task succeeded.

4.7(a): Tomato Cluster

4.7(b): Red Tomatoes Segmented

4.7(c): Green Tomatoes Segmented

4.7(d): Tomato Cluster



4.7(e): Red Tomatoes Segmented

4.7(f): Green Tomatoes Segmented

4.7(g): Tomato Cluster

4.7(h): Red Tomatoes Segmented

4.7(i): Green Tomatoes Segmented

Figure 4.7: Examples of satisfactory segmentation partial results.

Figure 4.8: Examples of unsatisfactory segmentation results for red and green tomatoes.

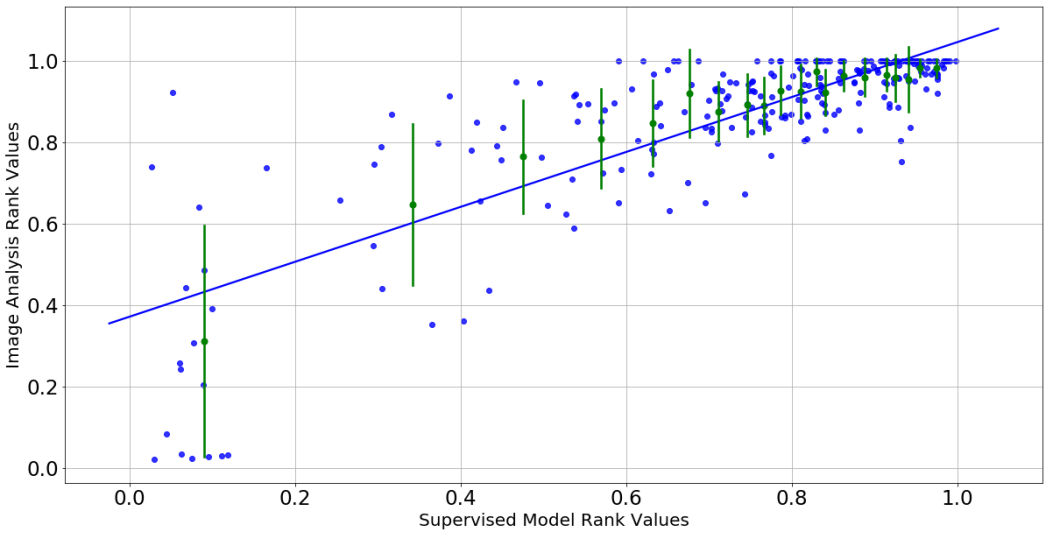


Figure 4.9: Scatter plot for maturation estimate. The x-axis represents the supervised rank estimate (rank_{sup}) and the y-axis represents the rank using image analysis (rank_{IA}).

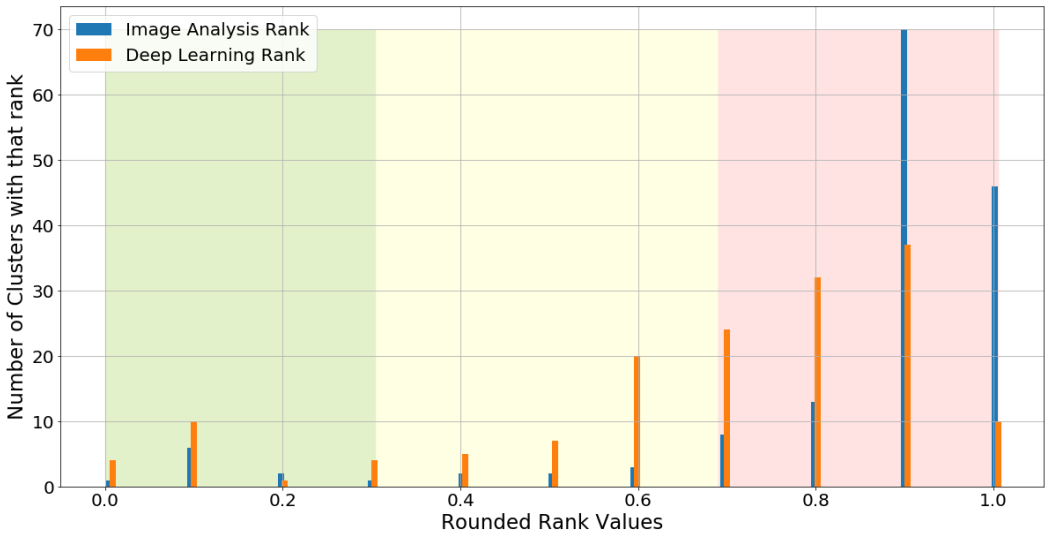


Figure 4.10: Comparative histogram for the DIT and DDT experiments for video 1.

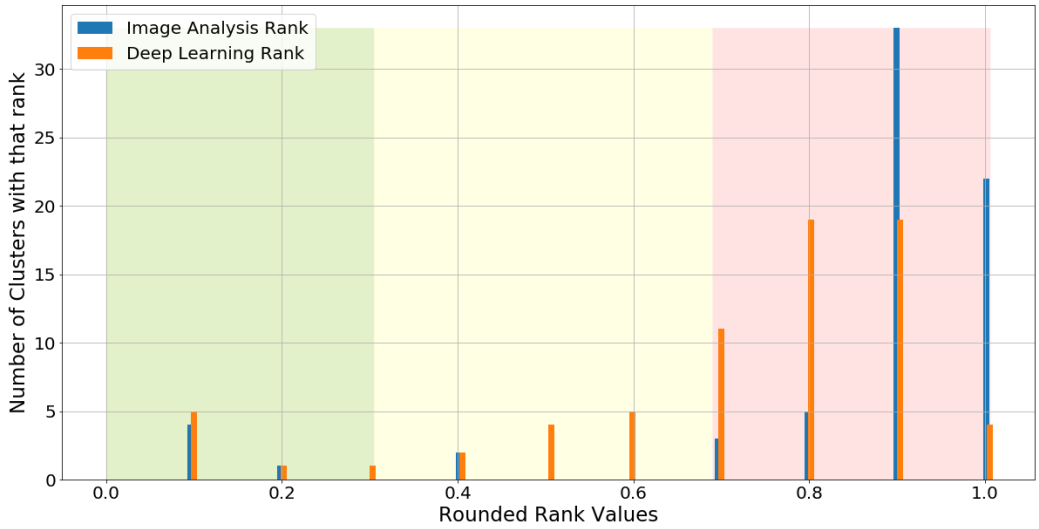


Figure 4.11: Comparative histogram for the DIT and DDT experiments for video 2.

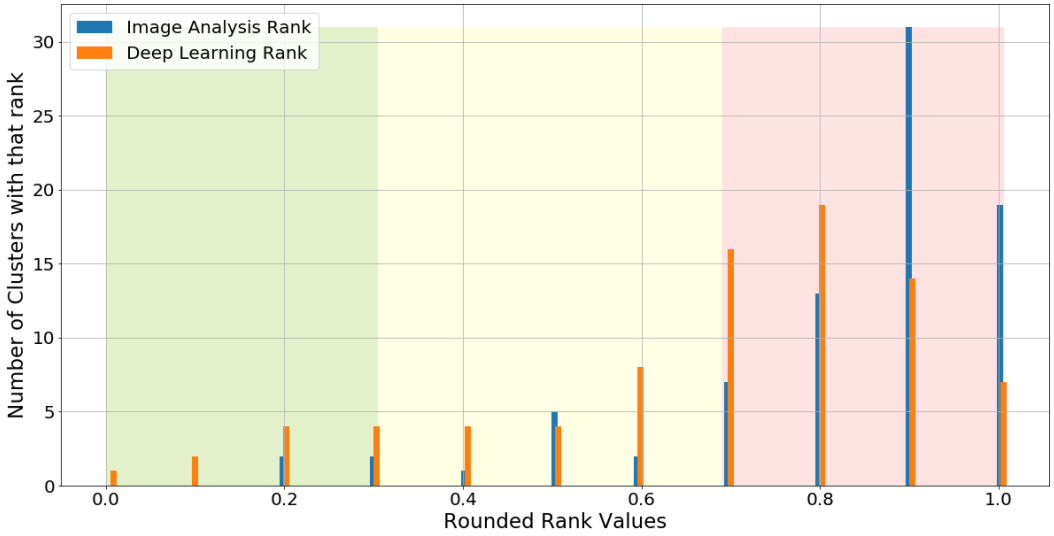


Figure 4.12: Comparative histogram for the DIT and DDT experiments for video 3.

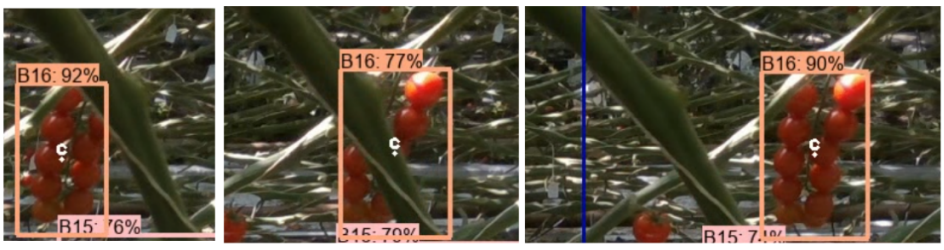


Figure 4.13: DDT results showing correct cluster detection in the presence of occlusion.

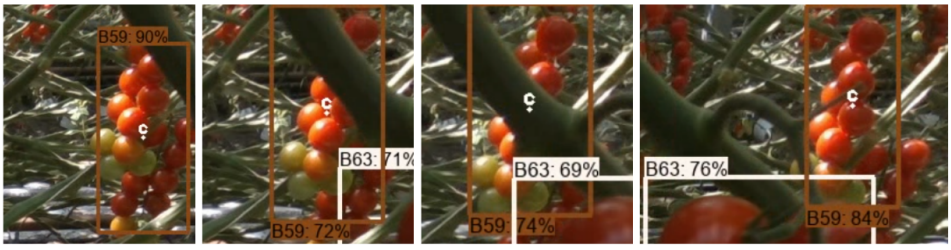


Figure 4.14: DDT results showing cluster rank discrepancy mitigation through the tracking algorithm.

Figure 4.15: DDT results showing loss and re-acquisition of a cluster by the tracking algorithm.

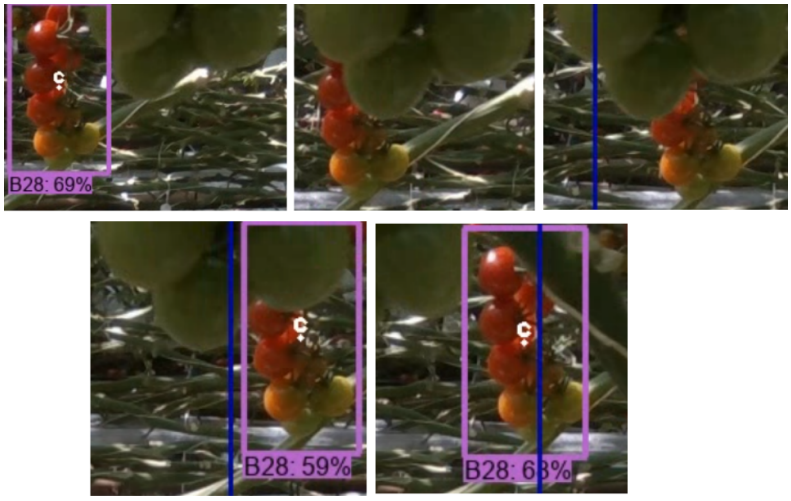


Figure 4.16: DDT results showing multiple losses and re-acquisition of a cluster by the tracking algorithm.

4.6 Conclusions and Future Work

In this paper, different approaches for fruit monitoring were implemented and tested proving its effectiveness. The supervised techniques showed the robustness to adverse conditions as well the precision of the convolutional neural networks for object detection and regression, when tuned properly. The object detection model could locate precisely the majority of tomato clusters on the test set. The baseline implementation based on segmentation was relevant to identify the tomato ripeness based on their appearance for clusters that contains mostly red tomatoes, but not so accurate for some mixed and green clusters. However, when using a CNN as a regressor for the same task, the maturity estimation was quite precise.

The counting system in videos based on region of interest is a simple and fast method to count objects, but it has limitations which lowers its precision. On the other hand, when the system was based on tracking, the algorithm was able to successfully identify the different clusters and consequently perform the counting with higher precision.

The processing speed, in general, is high and could help the farmer to identify the maturation quickly and with some consistency, improving the harvest efficiency. The system implemented could also be used together with ground robots for harvesting by informing the robot whether a cluster is ready for harvest or not. The robot may work for extended hours in different climates which would considerably accelerate production.

Training two separate supervised models (a detector and a regressor) has the advantage of being able to combine different architectures and easily observing the individual results as well the possibility to tune each one more precisely, but it is less efficient in terms of computational cost compared to a single model that performs both tasks. Besides the object location and class, the object detection algorithm can be adapted to provide the maturation rate by substituting the multiple classification layers for regression layers and also changing the loss function and the metrics. This approach was not considered in this paper because the VGG16 performed better for maturity estimate and MobileNetV1 perfomed better for cluster location, but the approach can be considered for other different CNNs architetures [21, 22, 23].

Also, the use of recurrent networks for instance segmentation could be tested for detection and tracking. Using supervised instance segmentation algorithms, it is possible to detect and segment each object sequentially and give different id for each one, being also robust to occlusion [81].

Instead of using regression or segmentation to estimate the maturation,

pixel-wise semantic segmentation deep learning algorithms could be used for this task, being robust to image adverse conditions (e.g. SegNet[9], U-Net [38] and DeepLab [37])

For future work, the tracking system could also be improved by using additional information to identify unique clusters such as the prediction of the next position of a given cluster based on adaptive Kalman filters, similar to the object tracking implementation by [82].

Acknowledgement

This work was partially funded by a Masters Scholarship supported by the National Council for Scientific and Technological Development (CNPq) at the Pontifical University Catholic of Rio de Janeiro, Brazil.

This work was partially supported by the UTFORSK Partnership Programme from The Norwegian Centre for International Cooperation in Education (SIU), project number UTF-2016-long-term/10097.

We thank the Applied Computational Intelligence Laboratory that provided the Nvidia ® GPUs and the Intel® Neural Compute Stick device at the Pontifical University Catholic of Rio de Janeiro, Brazil.

5
General Conclusions and Future Work

In this project, we presented the feasibility of supervised deep learning models in real applications for crop monitoring at three imagery levels. Both the literature and our own experiments show how the DL algorithms stood out in the areas of image recognition. We presented relevant results that can facilitate and improve the farmer's work environment leading to higher crop efficiency.

In the first sub-project, a newly proposed statistical analysis method and auxiliary mechanisms were implemented to improve the effectiveness of CNNs in transferring trainable parameters from different domains and learning the specific characteristics of vegetative species groups with high accuracy. The second sub-project used a semantic segmentation adapted model implementation, capable of providing multiple distinct but related information at once which is relevant for sugarcane crop analysis. The third sub-project uses object detection and CNN regression models to successfully locate tomato clusters and estimate their maturity. The counting system developed based on tracking was able to successfully identify the unique clusters and consequently perform the counting with high precision.

For future work we intend to integrate the UAVs and UGVs systems together which may improve the crop monitoring. The remote sensing UAVs can provide the crop health information while the UGVs can move to the intended location and identify the source of the problem (e.g. types of plants diseases and pest infestation).

It is intended to monitor the changes of the crop over time via remote sensing images and use this information to train models, such as CNNs with sequential approaches and recurrent neural networks (e.g. Long Short-Term Memory[83]) to predict the crop efficiency using images which makes it possible to take early decisions and manage resources such as irrigation and fertilization [84, 85].

Bibliography

[1] TENORIO, G.; VILLALOBOS, C.; MENDOZA, L.; COSTA DA SILVA, E. ; CAARLS, W.. **Improving transfer learning performance: An application in the classification of remote sensing data**. In: PROCEEDINGS OF THE 11TH INTERNATIONAL CONFERENCE ON AGENTS AND ARTIFICIAL INTELLIGENCE - VOLUME 2: ICAART„ p. 174–183. INSTICC, SciTePress, 2019.

[2] HORLER, D.; DOCKRAY, M. ; BARBER, J.. **The red edge of plant leaf reflectance**. International Journal of Remote Sensing, 4(2):273–288, 1983.

[3] MAHAJAN, U.; RAJ, B.. **Drones for normalized difference vegetation index (ndvi), to estimate crop health for precision agriculture: A cheaper alternative for spatial satellite sensors**. In: INTERNATIONAL CONFERENCE ON INNOVATIVE RESEARCH IN AGRICULTURE, FOOD SCIENCE, FORESTRY, HORTICULTURE, AQUACULTURE, ANIMAL SCIENCES, BIODIVERSITY, ECOLOGICAL SCIENCES AND CLIMATE CHANGE (AFHABEC-2016), AT JAWAHARLAL NEHRU UNIVERSITY, 2017.

[4] PRINCE, S.. **Satellite remote sensing of primary production: comparison of results for sahelian grasslands 1981-1988**. International Journal of Remote Sensing, 12(6):1301–1311, 1991.

[5] AITKENHEAD, M.; DALGETTY, I.; MULLINS, C.; MCDONALD, A. J. S. ; STRACHAN, N. J. C.. **Weed and crop discrimination using image analysis and artificial intelligence methods**. Computers and electronics in Agriculture, 39(3):157–171, 2003.

[6] IMRANAHMED, A. A.; ISLAM, M. ; GUL, S.. **Edge based real-time weed recognition system for selective herbicides**. In: PROCEEDINGS OF THE INTERNATIONAL MULTICONFERENCE OF ENGINEERS AND COMPUTER SCIENTISTS, volumen 1, 2008.

[7] BELAID, L. J.; MOUROU, W.. **Image segmentation: a watershed transformation algorithm**. Image Analysis & Stereology, 28(2):93–102, 2011.

[8] SINGH, V.; MISRA, A. K.. **Detection of plant leaf diseases using image segmentation and soft computing techniques**. Information Processing in Agriculture, 4(1):41–49, 2017.

[9] BADRINARAYANAN, V.; KENDALL, A. ; CIPOLLA, R.. **Segnet: A deep convolutional encoder-decoder architecture for image segmentation**. IEEE transactions on pattern analysis and machine intelligence, 39(12):2481–2495, 2017.

[10] GU, J.; WANG, Z.; KUEN, J.; MA, L.; SHAHROUDY, A.; SHUAI, B.; LIU, T.; WANG, X.; WANG, G.; CAI, J. ; OTHERS. **Recent advances in convolutional neural networks**. Pattern Recognition, 77:354–377, 2018.

[11] VAIPHASA, C.; PIAMDUAYTHAM, S.; VAIPHASA, T. ; SKIDMORE, A. K.. **A normalized difference vegetation index (ndvi) time-series of idle agriculture lands: A preliminary study**. Engineering journal, 15(1):9–16, 2011.

[12] GOODFELLOW, I.; BENGIO, Y. ; COURVILLE, A.. **Deep Learning**. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] LECUN, Y.; BENGIO, Y. ; HINTON, G.. **Deep learning**. nature, 521(7553):436, 2015.

[14] MICHELUCCI, U.. **Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks**. Apress, 2018.

[15] RUSSELL, S.; NORVIG, P.. **Artificial intelligence: A modern approach. third edition**, 2010.

[16] MOHRI, M.; ROSTAMIZADEH, A. ; TALWALKAR, A.. **Foundations of machine learning**. MIT press, 2018.

[17] NOGUEIRA, K.; DOS SANTOS, J. A.; FORNAZARI, T.; SILVA, T. S. F.; MORELLATO, L. P. ; TORRES, R. D. S.. **Towards vegetation species discrimination by using data-driven descriptors**. In: PATTERN RECOGNITON IN REMOTE SENSING (PRRS), 2016 9TH IAPR WORKSHOP ON, p. 1–6. IEEE, 2016.

[18] HE, K.; SUN, J.. **Convolutional neural networks at constrained time cost**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 5353–5360, 2015.

[19] PAN, S. J.; YANG, Q.. **A survey on transfer learning**. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2009.

[20] SIMONYAN, K.; VEDALDI, A. ; ZISSERMAN, A.. **Deep inside convolutional networks: Visualising image classification models and saliency maps**. arXiv preprint arXiv:1312.6034, 2013.

[21] SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V. ; RABINOVICH, A.. **Going deeper with convolutions**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 1–9, 2015.

[22] HE, K.; ZHANG, X.; REN, S. ; SUN, J.. **Deep residual learning for image recognition**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 770–778, 2016.

[23] KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 1097–1105, 2012.

[24] RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. ; OTHERS. **Imagenet large scale visual recognition challenge**. International Journal of Computer Vision, 115(3):211–252, 2015.

[25] CIREŞAN, D. C.; MEIER, U.; GAMBARDELLA, L. M. ; SCHMIDHUBER, J.. **Deep, big, simple neural nets for handwritten digit recognition**. Neural computation, 22(12):3207–3220, 2010.

[26] CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O. ; KEGELMEYER, W. P.. **Smote: synthetic minority over-sampling technique**. Journal of artificial intelligence research, 16:321–357, 2002.

[27] DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K. ; FEI-FEI, L.. **Imagenet: A large-scale hierarchical image database**. In: 2009 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 248–255. Ieee, 2009.

[28] REFAEILZADEH, P.; TANG, L. ; LIU, H.. **Cross-validation**. In: ENCYCLOPEDIA OF DATABASE SYSTEMS, p. 532–538. Springer, 2009.

[29] JOLLIFFE, I.. **Principal component analysis**. In: INTERNATIONAL ENCYCLOPEDIA OF STATISTICAL SCIENCE, p. 1094–1096. Springer, 2011.

[30] MAATEN, L. V. D.; HINTON, G.. **Visualizing data using t-sne**. Journal of machine learning research, 9(Nov):2579–2605, 2008.

[31] LIU, G.; GIFFORD, D.. **Visualizing feature maps in deep neural networks using deepresolve a genomics case study**. ICML Visualization Workshop, 2017.

[32] GOYAL, P.; DOLLÁR, P.; GIRSHICK, R.; NOORDHUIS, P.; WESOLOWSKI, L.; KYROLA, A.; TULLOCH, A.; JIA, Y. ; HE, K.. **Accurate, large minibatch sgd: training imagenet in 1 hour**. arXiv preprint arXiv:1706.02677, 2018.

[33] STEHLING, R. O.; NASCIMENTO, M. A. ; FALCÃO, A. X.. **A compact and efficient image retrieval approach based on border/interior pixel classification**. In: PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWLEDGE MANAGEMENT, p. 102–109. ACM, 2002.

[34] PASS, G.; ZABIH, R. ; MILLER, J.. **Comparing images using color coherence vectors**. In: PROCEEDINGS OF THE FOURTH ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, p. 65–73. ACM, 1997.

[35] SWAIN, M. J.; BALLARD, D. H.. **Color indexing**. International journal of computer vision, 7(1):11–32, 1991.

[36] UNSER, M.. **Sum and difference histograms for texture classification**. IEEE transactions on pattern analysis and machine intelligence, (1):118–125, 1986.

[37] CHEN, L.-C.; PAPANDREOU, G.; KOKKINOS, I.; MURPHY, K. ; YUILLE, A. L.. **Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs**. IEEE transactions on pattern analysis and machine intelligence, 40(4):834–848, 2018.

[38] RONNEBERGER, O.; FISCHER, P. ; BROX, T.. **U-net: Convolutional networks for biomedical image segmentation**. In: INTERNATIONAL CONFERENCE ON MEDICAL IMAGE COMPUTING AND COMPUTER-ASSISTED INTERVENTION, p. 234–241. Springer, 2015.

[39] LI, D.; LIU, Y. ; CHEN, Y.. **Computer and Computing Technologies in Agriculture IV: 4th IFIP TC 12 Conference, CCTA 2010, Nanchang, China, October 22-25, 2010, Part II, Selected Papers**, volumen 345. Springer, 2011.

[40] MARIN, F. R.; LOPES-ASSAD, M. L.; ASSAD, E. D.; VIAN, C. E. ; SANTOS, M. C.. **Sugarcane crop efficiency in two growing seasons in são paulo state, brazil**. Pesquisa Agropecuária Brasileira, 43(11):1449–1455, 2008.

[41] GUNNULA, W.; KOSITTRAKUN, M.; RIGHETTI, T. L.; WEERATHAWORN, P.; PRABPAN, M. ; OTHERS. **Normalized difference vegetation index relationships with rainfall patterns and yield in small plantings of rain-fed sugarcane**. Australian Journal of Crop Science, 5(13):1845, 2011.

[42] NICHOLSON, S.; FARRAR, T.. **The influence of soil type on the relationships between ndvi, rainfall, and soil moisture in semiarid botswana. i. ndvi response to rainfall**. Remote Sensing of Environment, 50(2):107–120, 1994.

[43] BORGOGNO-MONDINO, E.; LESSIO, A. ; GOMARASCA, M. A.. **A fast operative method for ndvi uncertainty estimation and its role in vegetation analysis**. European Journal of Remote Sensing, 49(1):137–156, 2016.

[44] GOLDEMBERG, J.. **Ethanol for a sustainable energy future**. science, 315(5813):808–810, 2007.

[45] SINGH, K. K.; SINGH, A.. **A study of image segmentation algorithms for different types of images**. International Journal of Computer Science Issues (IJCSI), 7(5):414, 2010.

[46] OTSU, N.. **A threshold selection method from gray-level histograms**. IEEE transactions on systems, man, and cybernetics, 9(1):62–66, 1979.

[47] ROSLAN, R.; JAMIL, N.. **Texture feature extraction using 2-d gabor filters**. In: 2012 INTERNATIONAL SYMPOSIUM ON COMPUTER APPLICATIONS AND INDUSTRIAL ELECTRONICS (ISCAIE), p. 173–178. IEEE, 2012.

[48] ROUSSON, M.; BROX, T. ; DERICHE, R.. **Active unsupervised texture segmentation on a diffusion based feature space**. In: 2003 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2003. PROCEEDINGS., volumen 2, p. II–699. IEEE, 2003.

[49] ZHANG, Z.. **A flexible new technique for camera calibration**. IEEE Transactions on pattern analysis and machine intelligence, 22, 2000.

[50] HEIKKILA, J.; SILVEN, O.. **A four-step camera calibration procedure with implicit image correction**. In: COMPUTER VISION AND

PATTERN RECOGNITION, 1997. PROCEEDINGS., 1997 IEEE COMPUTER SOCIETY CONFERENCE ON, p. 1106–1112. IEEE, 1997.

[51] BOUGUET, J.. **Camera calibration toolbox for matlab. computational vision at the california institute of technology**, 2012.

[52] LOWE, D. G.. **Distinctive image features from scale-invariant keypoints**. International journal of computer vision, 60(2):91–110, 2004.

[53] FISCHLER, M. A.; BOLLES, R. C.. **Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography**. Communications of the ACM, 24(6):381–395, 1981.

[54] CHUM, O.; PAJDLA, T. ; STURM, P.. **The geometric error for homographies**. Computer Vision and Image Understanding, 97(1):86–102, 2005.

[55] TANGSENG, P.; WU, Z. ; YAMAGUCHI, K.. **Looking at outfit to parse clothing**. arXiv preprint arXiv:1703.01386v1, 2017.

[56] IOFFE, S.; SZEGEDY, C.. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. arXiv preprint arXiv:1502.03167, 2015.

[57] IOFFE, S.; SZEGEDY, C.. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. arXiv preprint arXiv:1502.03167, 2015.

[58] KRIZHEVSKY, A.; SUTSKEVER, I. ; HINTON, G. E.. **Imagenet classification with deep convolutional neural networks**. In: Pereira, F.; Burges, C. J. C.; Bottou, L. ; Weinberger, K. Q., editors, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 25, p. 1097–1105. Curran Associates, Inc., 2012.

[59] BISCHKE, B.; HELBER, P.; FOLZ, J.; BORTH, D. ; DENGEL, A.. **Multi-task learning for segmentation of building footprints with deep neural networks**. arXiv preprint arXiv:1709.05932, 2017.

[60] KENDALL, A.; GAL, Y. ; CIPOLLA, R.. **Multi-task learning using uncertainty to weigh losses for scene geometry and semantics**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 7482–7491, 2018.

[61] KRESS-ROGERS, E.; BRIMELOW, C. J.. **Instrumentation and sensors for the food industry**, volumen 65. Woodhead Publishing, 2001.

[62] MŁOTEK, M.; KUTA, Ł.; STOPA, R. ; KOMARNICKI, P.. **The effect of manual harvesting of fruit on the health of workers and the quality of the obtained produce**. Procedia Manufacturing, 3:1712–1719, 2015.

[63] BREZMES, J.; LLOBET, E.; VILANOVA, X.; SAIZ, G. ; CORREIG, X.. **Fruit ripeness monitoring using an electronic nose**. Sensors and Actuators B: Chemical, 69(3):223–229, 2000.

[64] POVEY, M.. **Ultrasonics in food engineering part ii: Applications**. Journal of Food Engineering, 9(1):1–20, 1989.

[65] KONDO, N.; YATA, K.; IIDA, M.; SHIIGI, T.; MONTA, M.; KURITA, M. ; OMORI, H.. **Development of an end-effector for a tomato cluster harvesting robot**. Engineering in Agriculture, Environment and Food, 3(1):20–24, 2010.

[66] KONDO, N.; YAMAMOTO, K.; SHIMIZU, H.; YATA, K.; KURITA, M.; SHIIGI, T.; MONTA, M. ; NISHIZU, T.. **A machine vision system for tomato cluster harvesting robot**. Engineering in Agriculture, Environment and Food, 2(2):60–65, 2009.

[67] VIOLA, P.; JONES, M.. **Robust real-time face detection**. In: NULL, p. 747. IEEE, 2001.

[68] VIOLA, P.; JONES, M. J. ; SNOW, D.. **Detecting pedestrians using patterns of motion and appearance**. In: NULL, p. 734. IEEE, 2003.

[69] GIRSHICK, R.; DONAHUE, J.; DARRELL, T. ; MALIK, J.. **Rich feature hierarchies for accurate object detection and semantic segmentation**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 580–587, 2014.

[70] GIRSHICK, R.. **Fast R-CNN**. In: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION, p. 1440–1448, 2015.

[71] REN, S.; HE, K.; GIRSHICK, R. ; SUN, J.. **Faster R-CNN: Towards real-time object detection with region proposal networks**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 91–99, 2015.

[72] REDMON, J.; DIVVALA, S.; GIRSHICK, R. ; FARHADI, A.. **You only look once: Unified, real-time object detection**. In: PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, p. 779–788, 2016.

[73] LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y. ; BERG, A. C.. **Ssd: Single shot multibox detector**. In: EUROPEAN CONFERENCE ON COMPUTER VISION, p. 21–37. Springer, 2016.

[74] HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M. ; ADAM, H.. **Mobilenets: Efficient convolutional neural networks for mobile vision applications**. arXiv preprint arXiv:1704.04861, 2017.

[75] HECHT-NIELSEN, R.. **Theory of the backpropagation neural network**. In: NEURAL NETWORKS FOR PERCEPTION, p. 65–93. Elsevier, 1992.

[76] SZEGEDY, C.; REED, S.; ERHAN, D.; ANGUELOV, D. ; IOFFE, S.. **Scalable, high-quality object detection**. arXiv preprint arXiv:1412.1441, 2014.

[77] OHTA, Y.-I.; KANADE, T. ; SAKAI, T.. **Color information for region segmentation**. Computer graphics and image processing, 13(3):222–241, 1980.

[78] WEI, X.; JIA, K.; LAN, J.; LI, Y.; ZENG, Y. ; WANG, C.. **Automatic method of fruit object extraction under complex agricultural background for vision system of fruit picking robot**. Optik-International Journal for Light and Electron Optics, 125(19):5684–5689, 2014.

[79] BIANCO, S.; CADENE, R.; CELONA, L. ; NAPOLETANO, P.. **Benchmark analysis of representative deep neural network architectures**. IEEE Access, 6:64270–64277, 2018.

[80] ÖZLÜ, A.. **Tensorflow object counting api**, 2018. `https://github.com/ahmetozlu/tensorflow_object_counting_api`.

[81] ROMERA-PAREDES, B.; TORR, P. H. S.. **Recurrent instance segmentation**. In: EUROPEAN CONFERENCE ON COMPUTER VISION, p. 312–329. Springer, 2016.

[82] WENG, S.-K.; KUO, C.-M. ; TU, S.-K.. **Video object tracking using adaptive kalman filter**. Journal of Visual Communication and Image Representation, 17(6):1190–1208, 2006.

[83] GERS, F. A.; SCHMIDHUBER, J. ; CUMMINS, F.. **Learning to forget: Continual prediction with LSTM**. 1999.

[84] YOU, J.; LI, X.; LOW, M.; LOBELL, D. ; ERMON, S.. **Deep gaussian process for crop yield prediction based on remote sensing data**. In: THIRTY-FIRST AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2017.

[85] WANG, A. X.; TRAN, C.; DESAI, N.; LOBELL, D. ; ERMON, S.. **Deep transfer learning for crop yield prediction with remote sensing data**. In: PROCEEDINGS OF THE 1ST ACM SIGCAS CONFERENCE ON COMPUTING AND SUSTAINABLE SOCIETIES, p. 50. ACM, 2018.