Yield Estimation and Smart Harvesting for Precision Agriculture using Deep Learning

by

Youssef Osman

A thesis submitted to the
School of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Masters of Applied Science in **Electrical and Computer Engineering**

Faculty of Engineering and Applied Science
University of Ontario Institute of Technology (Ontario Tech University)
Oshawa, Ontario, Canada
August 2021

THESIS EXAMINATION INFORMATION

Submitted by: **Youssef Osman**

Master of Applied Science in **Electrical and Computer Engineering**

Thesis title: Yield Estimation and Smart Harvesting for Precision Agriculture using Deep Learning

An oral defense of this thesis took place on July 23, 2021 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Shahryar Rahnamayan
Research Supervisor	Dr. Khalid Elgazzar
Examining Committee Member	Dr. Ramiro Liscano
Thesis Examiner	Dr. Abdallah Shami, Western University

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

Precision agriculture is one of the fastest growing fields in recent years. In this thesis, we introduce a framework that provides farmers with a yield estimation from videos of crops and provides guided assistance for harvesting across the farm by utilizing geospatial information that is collected during the recording of the crops. We perform yield estimation by using a tracking model, DeepSORT, that can keep track of detected fruits for accurate counting. We modified the original DeepSORT algorithm to work efficiently on different fruits without the need for retraining. The proposed framework also provides assistance for smart harvesting through an optimized approach for container placement across the field. Performance evaluation shows that the proposed method achieves more than 90% accuracy on a real video footage of apple trees collected by a drone from an apple orchard and approximately 94% accuracy for pumpkin counting from an aerial drone footage.

Keywords: precision agriculture; deep learning; computer vision; geospatial data; agriculture decision support

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Youssef Osman

Statement of Contributions

The contributions that accompany this thesis include an accepted paper in the 2021 IEEE World Forum on Internet of Things (WF-IOT 2021), a submitted paper in IEEE ACCESS, and a submitted journal paper to Sensors. This is described in more detail below.

1. Referred Conference Proceedings:

Youssef Osman, Reed Dennis and Khalid Elgazzar, "Yield Estimation Using Deep Learning for Precision Agriculture," in Proc. IEEE World Forum on Internet of Things, New Orleans, Louisiana, USA, 2021. In this paper, the author introduced a deep-learning based pipeline for yield estimation from video feed of fruits. The pipeline included two stages: object detection and object tracking. For object detection, YOLOv3 model was selected due to its balance between accuracy and inference speed. The model was trained on images of apples. The author introduced an annotation strategy when preparing a fruit dataset, that allows the model to learn different visual challenges. In the next stage, DeepSORT algorithm was used for tracking. The algorithm uses deep learning features that were only trained on people, thus the authors modified this module to work on various fruits without needing to train the tracking module. Results were presented and showed the effectiveness of the pipeline, and how training YOLOv3 with the highlighted strategy improved its performance. This paper has been successfully accepted and presented at IEEE WF-IOT 2021. This paper is based on the yield estimation framework presented in Chapter 3, the annotation strategy is also used and shown in Chapter 4, and the results are included in Chapter 4.

2. Referred Conference Proceedings:

Reed Dennis, Youssef Osman, Sifatul Mostafi and Khalid Elgazzar, "Quantitative Analysis of Deep Learning Object Detection Models," in IEEE ACCESS. In this paper, the authors present a comprehensive comparison between popular deep learning based object detection models. Each model is briefly explained, and tested on the COCO

dataset. A comparison between each model's accuracy, and inference speed is highlighted. This paper is submitted and under review at IEEE ACCESS. This paper covers numerous aspects of background studies in object detection, which is presented in Chapter 2.

3. Referred Journal Proceedings:

Youssef Osman, Reed Dennis and Khalid Elgazzar, "Yield Estimation and Visualization Solution for Precision Agriculture," in Sensors. This paper is an extension of the previous paper "Yield Estimation Using Deep Learning for Precision Agriculture". In this paper, the authors further expanded their experimentation on the pipeline to other fruits to test its versatility. The authors also perform experiments on full rows of apple trees. Once the yield is estimated for the apple tree rows, the authors combined the count with geospation data to be used for visualization. Once the yield is visualized, the authors introduce an optimal container placement solution, that suggests a number of containers and locations to place them based on the previously processed yield data. This paper is submitted and under review at Sensors. This paper consists of the full framework that's presented in Chapter 3, and its results presented in Chapter 4.

Acknowledgements

I would like to thank many people for their support. It's been an extremely challenging period in my life, and I couldn't have made it to the finish line without the help, guidance and support of others.

First and foremost, I would like to thank my supervisor, Dr. Khalid Elgazzar. Dr. Khalid has been extremely patient with me, constantly listened to all my ideas, but made sure to keep me in track and turn my ideas into practical and functional work. His guidance has been imperative in my success, and my gratitude for his supervision is indefinite.

I would also like to thank my lab-mates within Dr. Khalid's Internet of Things research lab. I could not have asked for better people whom each contributed in a meaningful way. From criticising my work, to explaining some concepts I hadn't learned yet, to constantly cheering me on through every progress update I gave, they've all been true friends. I can only hope be as encouraging and supportive to them as they were to me.

Lastly, and most importantly, I would like to thank my family. There aren't enough words I could use to describe how blessed and thankful I am for my parents and brothers. They've supported me through every single step I've taken in this life, and always made sure they could give me the best. Being thousands of miles away, and they still give all the support they physically can. I owe it all to them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Precision agriculture is an ever-growing domain where technology in its various forms is used in different facets of agriculture. This includes using a wireless sensor network to monitor the soil conditions across an entire field of plants, flying drones that dispense water and nutrients to fields of crops, and autonomous robots that can navigate through a field and harvest the ripe fruits and vegetables. The uses of data analytics, intelligent sensing, robots and other modern technologies in agriculture are endless. Incorporating precision agriculture brings numerous benefits to farmers, such as: 1) autonomously handling irrigation, fertilization and treatments that saves money and efforts and done more efficiently, 2) identifying regions that are affected by weeds or diseases and isolate them quickly which saves the rest of the field from harm, 3) localizing areas with healthy and productive soils, to allow for efficient crop distribution, 4) calculating packaging and production costs based on yield estimation for accurate logistics planning, 5) measuring the ripeness of the crop to set up precise harvesting schedules. Having various types of information about the field, soil, and crops directly aids the farmer in making informed decisions, and positively impacts profitability. For example, Corwin et al. [9] used soil sensors to investigate the effects of electrical conductivity of the soil on its productivity and found a correlation where conductivity highly influences the efficiency of the soil. Tian et al. [59] focused on addressing weed problems within crops by introducing a weed detection solution. They use sensors that can detect weeds and inform the farmer of potential weed areas so that they may be immediately isolated and treated before spreading to other areas. There are numerous other areas where technology can be integrated into the agriculture process to make it smarter, efficient, and more productive. In this thesis, we specifically target using computer vision and deep learning techniques for yield estimation and harvesting optimization to support farmers in making informed decisions.

1.2 Motivation

Yield estimation (a.k.a fruit counting in this thesis) is the process of providing an estimated yield count of the crop. Yield estimates are rather vital to the harvesting process as they allow the farmer to make informed decisions [16]. Being informed of the yield facilitates planning efficient harvest routes, procuring equipment, assigning labor, and preparing for packaging and production. For example, orange yield estimates are necessary for orange juice manufacturers as orange juice must be made within 48 hours of the fruit's harvest time. Yield information is used in ensuring that juice plants are run optimally, at maximum capacity, within the manufacturing window [40]. Traditionally, fruit counting can be done by humans who manually count through the fields, or estimate based on historical data [45]. However, these techniques are not only labor intensive and time consuming, but also fairly inaccurate, with the latter being particularly susceptible to bias. Precision agriculture introduces a potential solution that avoids human error and provides relatively quick and accurate estimations. Specifically, the use of image processing provides the means to perform yield estimates on crops that are visible (e.g., apple trees, oranges, pumpkins) [43].

Many works that we preview in chapter 2.4 introduce various solutions for counting fruits and vegetables of all kinds, especially newer frameworks that incorporate artificial intelligence (AI), and specifically deep learning (DL) models. For our thesis, we develop a fully DL-based framework to perform yield estimation and support harvest decision making. We divide our framework into two main components: yield estimation and yield mapping. We perform yield estimation by counting fruit yield through videos. That way, our solution is robust enough to work on any type of fruit from video feeds as opposed to limited static images. This is quite efficient and scalable with the integration of mobile sensors (e.g., robots and drones) that would autonomously navigate through crop fields and capture video footage. We then incorporate spatial information about the yield to recommend optimal placement of harvest containers to reduce the number of required

bushels and save their collection efforts, enabling farmers to make better use of their resources while optimizing the harvesting process.

1.3 Problem Statement

Yield estimation is a challenging problem with multidimensional aspects. Specifically, computer vision techniques that provide fruit counts face numerous limitations. Classic computer vision techniques, that focus on analyzing pixel colors or brightness, rely heavily on fruits maintaining consistent colors. In addition, these techniques are incapable of generalizing to different fruit types. This limitation alone severely affects the practicality of classical techniques, as real fruit fields are filled with varying effects (such as sunlight exposure and tree densities). Modern solutions in yield estimation utilize deep learning. DL models are vastly more applicable as they are capable of generalization. In the context of fruits, some visual challenges include a fruit being partially occluded by a leaf or another fruit, similar fruits with varying color and texture features. DL models can learn such visual variances, however, current techniques do not explicitly express how. Moreover, as we perform fruit counting on video feeds, current techniques only function on still images and would fail to accurately count on a frame-by-frame basis.

In our thesis, we define the challenges that we address in our solution as follows:

1. Fruits are subject to various image quality issues that affect their visual appearance. In most fruit fields, visual feature challenges primarily include:
   Occlusion: where a leaf or fruit can intersect with, or heavily cover another fruit;
   Lighting conditions: where direct sunlight, or lack of thereof can change the color appearance of the fruit;
   Varying textures: subtle markings or differing color patterns can exist between the same fruits (e.g., some red apples can have yellowish colors when not fully ripe).

2. Traditional techniques that rely on a specific fruit's features struggle and can even

fail in these cases. For example, many existing techniques in apple counting design a detector that analyzes pixels within an image and classifies clustering red pixels as an apple. While functional, a method that focuses purely on color is bound to fail in practical scenarios where apples are within dense trees with many leaves that either cover the apple, or block sunlight and turn the red pixels of apples into much darker shades. Additionally, existing DL- based techniques don't explicitly propose an approach to preparing a dataset for visual variances.

3. Deep learning-based solutions provide robust detection models that can apply to fruits. However, fruit detection is insufficient to deduce a count in video format, detections are made on a frame-by-frame basis with no way to tell whether a detection in the first frame is the same as a detection in the second frame. This makes accurate counting impossible, as if a specific fruit "x" is detected in frames 1, 2, 3 and 4, it will be counted four times since there is no association between each time it gets detected.

4. Videos capturing for crops can contain varying lighting changes, or unexpected camera movements from encountering a bump, for example when a ground robot drives over fallen branches. Such a variation introduces challenges for the visual appearance of the same object between different frames. The object moves unexpectedly, and if we use a simple tracking algorithm based purely on the motion of the object, the sudden change in motion direction or speed will cause the algorithm to struggle.

5. Data is not provided in a digestible manner. Ultimately yield estimation is performed to aid in farmer decision making. While a total yield is helpful, it doesn't tell the farmer fruit fields tend to be massive, and a breakdown of where the yield is

1.4 Thesis Contribution

Our main contributions in this thesis are as follows:

1. A comprehensive yield estimation approach based on the latest deep learning technologies to transform the future of precision agriculture. We demonstrate the feasibility of this approach using three use cases of fruit counting.

2. A modified version of DeepSORT tracker using ResNet that can work with any ImageNet object with no need for retraining. This makes the proposed approach robust and fruit-independent. ResNet is robust, well researched, and comes in different variations to opt for faster speed or more accuracy.

3. A fully annotated apple dataset for research purposes with guidance on important considerations to make when annotating fruit datasets for yield estimation.

4. A visualization approach for projecting yield estimations on a map using geospatial points.

5. An approach for optimal container placement to support smart harvesting decision making, leveraging geospatial information combined with the yield analysis and opening up the potential for numerous decision support applications.

1.5 Thesis Organization

This thesis is organized as follows. Chapter one introduces yield estimation, the gaps and challenges in computer vision-based techniques, how we address these challenges and how we contribute. In chapter two, we provide a comprehensive background study on computer vision solutions in the context of deep learning, we perform a survey of object detection and tracking techniques, as well as frameworks for fruit counting related to this thesis. We present our end-to-end framework for yield estimation and visualization

in chapter three and propose a technique for optimal container placement in fruit fields. In chapter four, we overview our dataset preparation, propose a strategy for annotating apple images, and present our framework results in three fruit use cases (apples, oranges, and pumpkins). Lastly, we conclude with chapter five, providing some insight into our methodology, how it is successful and its potential limitations, and where our work can be extended in the future.

Chapter 2

Background and Related Work

2.1 Introduction

In this chapter we review the literature surrounding fruit analysis via computer vision. Specifically, with the rise of deep learning solutions and the presence of numerous state of the art techniques for crop classification and detection [45] [1][51] [7][40] [26][3][69][60][66]. We mainly focus on reviewing deep learning techniques for computer vision applied in precision agriculture to outline the state-of-the-art and identify the research gaps. We begin by providing some necessary background about object detection and tracking using different deep learning techniques. Then, we review the literature for related work.

2.2 Deep Learning in Computer Vision

The use of deep learning in computer vision has been the standard since the conception of Convolutional Neural Networks (CNNs). CNNs are a type of deep neural networks that are architected in a way that's optimal for the analysis of an image. AlexNet [18] revolutionized the application of deep learning for computer vision by providing a model capable of accurate classification within the massive ImageNet dataset consisting of 15 million images and 22 thousand classes. This shows that CNNs are capable of learning and distinguishing between thousands of classes using supervised learning (training the model on the annotated dataset that contains the image and its respective classification ground truth). It's worth noting that feature selection is completely autonomous in CNNs, all that's provided is the annotated dataset during the learning phase and the CNN is completely responsible for any feature processing and learning.

A typical CNN consists of multiple layers (mainly convolution and pooling operations that we will expand on in the following section), with a final fully connected layer that then makes a classification decision. In this section, we will delve into the concept of CNNs and look into how each layer functions.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks that, while specializing in image processing, is used as a deep learning solution for different types of classification tasks (such as signals and time-series analysis). CNNs were conceptualized after the rise and success of machine learning, and more specifically, Artificial Neural Networks (ANN) in prediction and classification tasks. ANNs are modeled in the form of a network divided into layers, with each layer containing nodes and interconnections between them. A simple neural network would consist of 3 layers, one input layer, one middle layer and one output layer. When there is more than one middle layer, the model can be considered a deep learning solution. Each node consists of a mathematical function that can be interpreted as some piece of information that would lead to the final prediction. The function is constructed as follows: $y = activation(W.x + b)$, where y is the output value of the node, activation is the activation function, W is the weight matrix of the connection, x is the input from the connection, and b is the bias. An activation function is a mathematical formula that's applied in order to see whether a node should "activate" or not. For example, after computing the value of the node, we can then apply the Sigmoid activation function that can transform the value into some number between 0.0 and 1.0. This is extremely helpful because it squishes the number to a value that's easier to process. Values closer to 0.0 mean the node should be deactivated and values closer to 1.0 mean the node should be activated. The weight matrix of the connection is a collection of values that express the importance of the input to the output. Meaning connections with high values of weight tend to mean that features within this connection are significant to identifying the overall output and vice versa. Weights are also considered the training values. To expand, what makes ANNs so efficient is their ability to identify patterns and learn. ANNs do so by going through a learning process where it first attempts to make a prediction, compares it with the actual results, and computes an error, which signifies the difference between the actual and predicted values. It then

adjusts the values of the weights in order to minimize the error, so that when it attempts to predict again, the prediction should be more accurate. This concept applies to further developments in neural networks, including CNNs.

CNNs expand on neural networks by introducing the convolution operation to the neural network structure. By inputting a 2-dimensional matrix, CNNs use filters (which are small matrices that contain the weight values in the context of CNNs) to slide over the matrix, one window at a time, and compute the dot product. This process is essentially convolution: a mathematical operation on two functions that produces a third function which describes how the shape of one is modified by the other represented by Eq. 2.1.

$$s(t) = \int x(a)w(t-a)da \tag{2.1}$$

In the context of CNNs, the terminologies are denoted as follows: the function x is the input and the function w is the kernel (also known as filter) with s being the feature map. When convolution is applied in CNNs, it's better described as multidimensional discrete convolution, as our inputs are multidimensional matrices, as is our output. Thus Eq. (2.1) is re-written as

$$S(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) \equiv S(i,j) = \sum_m \sum_n I(i-m,j-n)K(m,n) \tag{2.2}$$

A visual representation of the convolution operation within a CNN can be found in Fig. 2.1. In the CNN architecture, convolution operations are performed within the convolutional layers. These layers are the backbone of CNNs and are responsible for extracting features from the input. The process follows the aforementioned equation and is as follows: 1) a kernel is placed over a certain point of the input, 2) each value in the kernel is multiplied by its respective value in the current input window, 3) the summation of the computed values is calculated, 4) the kernel slides over to the next window based on a stride value (e.g., if the stride is 1, the kernel shifts to the left by 1 index point) and

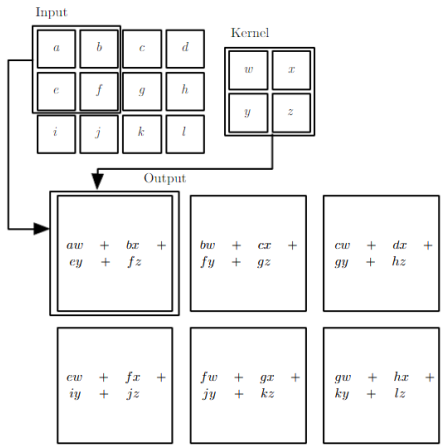


Figure 2.1: Example of basic convolution in CNNs [17].

repeats the same operation until it covers all input values. Theoretically, convolutional layers enable the extraction of multiple levels of features. For example, when analyzing a picture of a cat, the first convolutional layer can extract the fur of the cat, the second layer can extract the color information, and the third layer can focus on physical features and so on. As the network grows deeper (i.e., use of numerous convolutional layers), many features with increasing complexities are extracted, leading to an overall improved performance.

The reason why CNN introduces the concept of kernels and convolution operations is because using the same kernels across the entire input essentially allows for sharing a smaller set of weights and feature detection across the entire image. This is known as parameter sharing. In a standard deep neural network, every connection between each pair of nodes has its own weight value. However, with CNNs, kernels contain the set of weights that we slide over the entire image, leading to the model requiring significantly less memory space to store the learned weights, without affecting the run time or compromising how deep neural networks analyze and learn. Two visual examples of parameter sharing can be found in Figures 2.2-2.3.

The output of the convolutional layer is a feature map, which is essentially the analyzed features of the input. In some cases, the output matrix may need to be down-

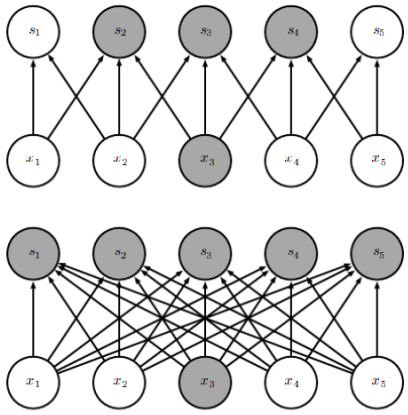


Figure 2.2: The concept of parameter sharing, as illustrated when a convolution is performed with a kernel of width 3, only three outputs are affected by one input. This is opposite to the bottom image where convolution isn't performed and all outputs are affected by a single input [17]

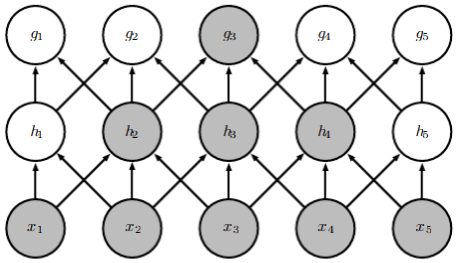


Figure 2.3: An example of the receptive field as a convolutional neural network grows deeper. This displays that even though direct connections in CNNs are sparse, nodes in deeper layers can be indirectly connected to all or most of the input image [17]

.

sampled, whether to speed up following convolution operations, or simply to produce a more efficient feature map. Pooling is responsible for reducing the size of the matrix by grouping each set of pixels together while preserving as much information about the features as possible. This is feasible due to the fact that with parameter sharing, and the nature of images, closely neighboring pixels may contain information similar to each other. After convolution, this similarity may lead to some redundancy within the output feature map which pooling can remove. There are different functions that can be used for pooling, but typically either max pooling or average pooling is performed. For max pooling, within each window of the feature map, the maximum value is selected to represent

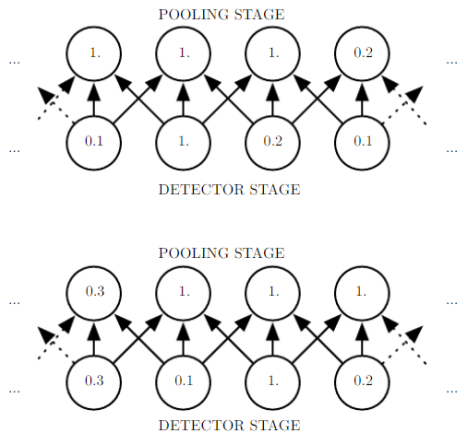


Figure 2.4: An example of how a max-pooling layer works [17].

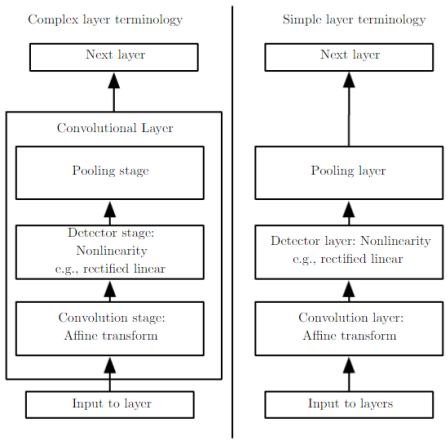


Figure 2.5: An example of a generic convolutional network layer. [17].

the feature of this window. This concept also applies to average pooling, except with the average value being computed instead. An example of a generic convolutional network layer can be seen in Fig. 2.5. The final step within a CNN is to input the finalized feature map into a fully connected deep neural network, known as the fully connected layer. This layer behaves like a standard neural network and is responsible for outputting the final prediction.

2.2.2 Implementation of CNN Architectures

With the advancements in CNNs, there has been numerous successes in developing powerful CNN models for feature extraction and image classification. These models are incredibly versatile because they're trained on the huge ImageNet dataset. Nowadays, it's standard practice to refer to such models when working with CNNs as they're well established in research and saves a lot of effort when it comes training as users can use transfer learning for their specific application. Transfer learning is a form of learning where existing weights (such as weights that are pretrained on ImageNet) are frozen and used as a starting point for the training process, instead of training from scratch. It has been proven that transfer learning is a viable and efficient strategy when dealing with CNNs [24]. In this section, we examine one of the earliest, most successful, and most used image-recognition CNN architecture, VGG [54].

VGG

After the success of AlexNet, many works followed similar formulas in order to develop new and better CNNs. VGG [54] achieved success by focusing on the primary point of improvement for CNNs: the depth of the network. One of the architectures of VGG that takes an input size of 224 x 224. The general filter sizes are 3 x 3, with some filters being 1 x 1 to act as linear transformations. Max pooling is used in all pooling layers. Following the pooling layers, there's a fully connected layer that consists of 2 levels having the size of 4096 (the output size of the final pooling layer), and the last level having 1000 nodes (one for each class in the ImageNet competition). It was found that by using small filters, there's a large number of weights that lead to more layers and improved accuracy. Since VGG, it has been staple to try to make the CNN as deep as possible, but try to find ways to preserve fast and efficient performance as well.

2.3 Object Detection

A standard CNN architecture alone is insufficient for more complex computer vision tasks. To be specific, further processing is needed to solve one of the most popular, and most in-demand computer vision problems, object detection. Object detection is the process of detecting objects (e.g., people, cars, food, etc.) and classifying them. Generally, the detections are output in the form of bounding boxes that possess pixel-wise coordinates and surround each object analyzed within an image. An object detection solution is vital to our work to analyze our images of fruits and count the detections to produce a yield estimate. Fortunately, nowadays there is a plethora of deep learning-based object detection algorithms that expand on CNNs. Object detection models leverage CNNs to analyze the image and incorporate other CNN components such as convolutional and pooling layers to localize objects within the image and output their pixel-wise location and classification. The biggest advantage of the deep learning approach to object detection is their performance and ability to generalize, while avoiding the complexity of manually engineering features. Feature engineering is sometimes challenging to design and represent in a way that computers can practically understand. In this section, we explore various state-of-the-art deep learning-based object detection models.

2.3.1 Fast and Faster R-CNN

Fast R-CNN [13] is one of the earliest "real-time"[1] object detectors that used region-based detection, namely R-CNN [14]. The general idea is to propose multiple regions of interests across the image and then search within these regions for objects. Fast R-CNN focuses on the latter, designing a Region of Interest (RoI) layer for analyzing potential regions of interests and detecting the objects within them.

The model takes an image and potential regions of interest as inputs and begins the

[1] At the time that Fast R-CNN was published, the detection network was able to achieve inference speeds of 0.3s per image. Which was better than any other model to date.

process of object proposals. A feature map of the image is first extracted by passing the image through the backbone network until the fully connected and classification layers. Afterwards, the regions of interest are extracted from the feature map of the entire image, producing several feature maps, each belonging to a specific region. The model applies a max pooling operator to every extracted region's feature map. These feature maps are downsized to fixed length feature vectors. These new feature vectors are smaller and could be quickly analyzed while still maintaining important features of the region. The output of the RoI layer is passed to two separate output layers. The first layer classifies the object within the region while the second regresses the bounding box coordinates.

Faster R-CNN [49] expands on Fast R-CNN by introducing a CNN based region proposal network (RPN) that's responsible for predicting region proposals. These region proposals are used by the Fast R-CNN architecture[13] to produce class probabilities and bounding boxes. The RPN introduced takes a feature map produced by a convolutional layer and generates a set of RoIs, each with their own "objectness" score. An objectness score of a RoI estimates the likelihood that it contains an object. The model first propagates an image through the backbone network, producing a feature map which will subsequently generate the region proposals via the RPN. A sliding window then moves across said feature map, extracting the feature and transforming them into a fixed length feature vector.

Faster R-CNN then passes the feature vector to two separate fully connected layers, where the first layer regresses the bounding box coordinates and the second classifies the objects contained within the region proposal. During this process, an anchoring operation occurs for each window of the original feature map. To elaborate, instead of taking just the single window with its fixed size, multiple boxes (also known as anchor boxes) are used as priors for the bounding box regression.

Anchor boxes have different sizes and aspect ratios (For example, one box could be a square shape, others could have rectangular shapes with different dimensions). All

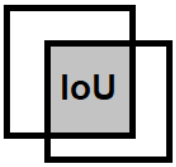


Figure 2.6: Intersection over Union shows how much two objects overlap with each other, with 1.0 denoting that two objects fully overlap and 0.0 means that the two objects have no overlap

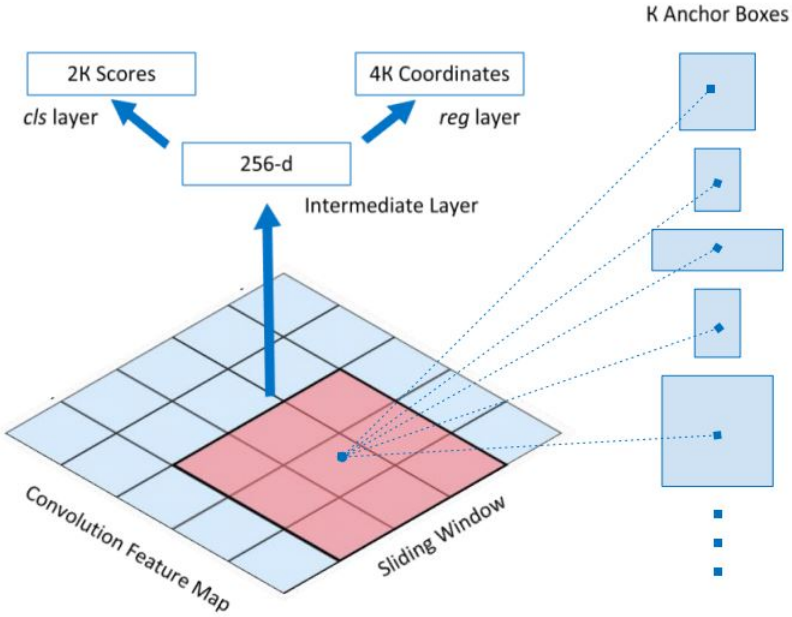


Figure 2.7: Anchor box mechanism in Faster R-CNN. The mechanism is adapted in future object detectors as well, such as YOLO, reproduced from [49].

predicted boxes undergo the same two following layers, and the anchor boxes with the highest probability and the highest intersection over union (IoU) as shown in Fig. 2.6 are chosen. We show this layer in Fig. 2.7. It's important to note that the output of the RPN are region proposals, not detections, thus the output regions of interest used as input for the Fast R-CNN model which performs object detection and classification.

2.3.2 Mask R-CNN

Mask R-CNN [19], an extension of Faster R-CNN, was designed for instance segmentation tasks. In computer vision, the goal of a segmentation task is to produce a segmentation mask where each pixel belongs to a specific class. Consider an image of a cat standing on grass, each pixel can either belong to the cat class or the grass class. The segmentation maps pixels belonging to the cat with a value of 0 whereas the pixels belonging to the grass class would take a value of 1. Instance segmentation refers to performing segmentation on specific instances within the image as opposed to the entire image. Mask R-CNN achieves this by performing object detection, then segmenting the image within each detection box.

The architecture of Faster R-CNN, examined in the previous section, is adopted in Mask R-CNN. As highlighted in Fig. 2.8, the RoI layer extracts the feature map of every proposed region of interest within the image using a ResNet backbone. They pass the feature maps to a fully connected layer where objects are classified and their bounding box coordinates are regressed. The primary difference in this model is that it performs segmentation via a separate fully convolutional branch alongside the bounding box regression. Extracted RoI feature maps are forwarded into a FCN, similar to the work done in [53] for semantic segmentation. Thus, the final output comprises the detected object bounding box, its classification, and the mask within the bounding box. The addition of the mask allows Mask R-CNN to be an all-around framework capable of fulfilling instance-level tasks such as analysis of microscopy images .

One-Stage Detectors (Regression/Classification)

2.3.3 SSD

The Single Shot MultiBox Detector [37] (SSD) was among the first to use a pyramidal feature hierarchy approach for generic object detection. The main idea behind this

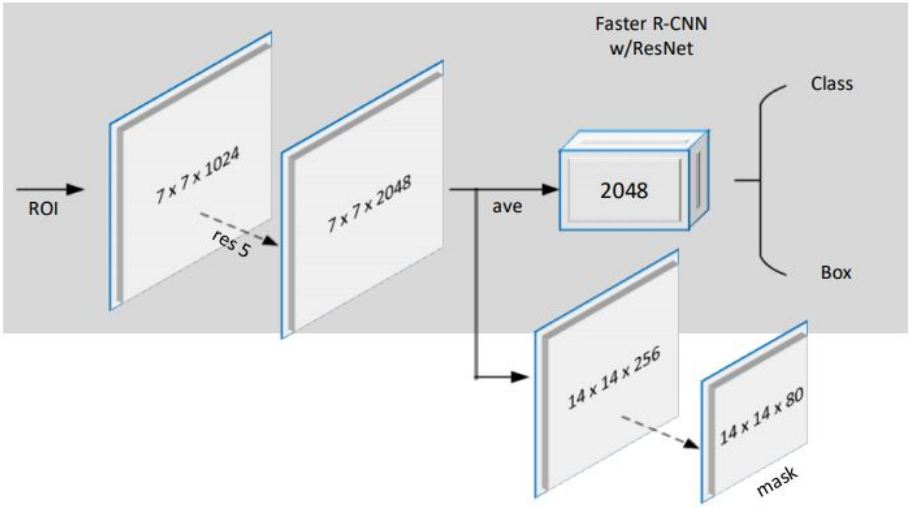


Figure 2.8: Mask R-CNN is divided into the detector, based on the Faster R-CNN model, and a layer that produces the mask of the input, reproduced from [19].

approach is to take a backbone network [2] and add extra convolutional layers that are responsible for both bounding box classification and regression using the features produced from various feature maps of different scales. As such, these added layers progressively decrease in size; high to low resolution feature maps. The higher resolution feature maps are responsible for detecting smaller objects while lower resolution feature maps are responsible for detecting larger objects. Each feature map chosen to be used as input for the regression/classification layer have k anchor boxes assigned to each feature map cell, e.g., a feature map of size 38×38 would have $38 \times 38 \times k$ anchor boxes in total. It is worth noting that the number of channels does not impact the number of anchor boxes. It solely relies on a predetermined number of anchor boxes and the $W \times H$ of the feature map. In [37] 8732 anchor-box based detections are produced per class in total.

[2] VGG-16 [54] is used in the paper, however, any backbone network is feasible as the approach is agnostic to the used backbone architecture.

2.3.4 YOLO, YOLOv3 and YOLOv4

The "You Only Look Once" (YOLO) class of object detectors was first introduced by Redmon et al. [48] in 2016. YOLO treats object detection as a regression problem of objects and their class probabilities separated into grid-spaces. Operating within the bounds of a single network, there is no region proposal network similar to it. As a result, YOLO can perform object detection at real-time speeds.

These real-time speeds are achieved by dividing an image into a two-dimensional grid, where the grid size is a hyper-parameter (typically 7x7). Every grid cell regresses a pre-defined number of bounding boxes and confidence scores, which is represented by the product between the probability that a grid cell contains an object and the intersection over union (IoU) of the ground-truth box and predicted box. Every bounding box contains five regressions, the center coordinates of the bounding box, the width and height, and the confidence. Determination of the bounding boxes class is done using the class probability map, which assigns every grid cell a set of class probabilities. The class chosen is often the one with the highest class-specific confidence score.

As with all object detectors, YOLOv3 needs a backbone to obtain a feature map of the input image. Unlike other models, however, YOLOv3 continues to use the DarkNet architecture, as it presents performance consistent with popular models often employed for feature extraction [47], but with fewer floating-point operations. Another notable addition presented by YOLOv3 is the multi-scale object detection, where instead of using the feature map extracted after the final convolutional operations, they used the last three stages of the DarkNet53 backbone for predictions akin to [31], with feature map dimensions of 32x32, 16x16 and 8x8.

Using multiple outputs from different stages improves detections, as larger objects are easier to detect in later stages, whereas smaller objects are better detected in earlier stages. They predict anchor boxes for the extracted feature maps, comparable to Faster R-CNN, keeping the boxes with the strongest confidence scores. An issue that arises is

that multiple bounding boxes can be regressed for a single object. To address this, all predicted boxes are sorted by the confidence score and the IoU between the box with the highest confidence score and all other bounding boxes are computed. If the IoU is above a certain threshold, it discards the other bounding box. This process, which is called non-maximum suppression, is applied iteratively to all bounding boxes within an image.

YOLOv4 uses the same anchor-based detection as YOLOv3 and focuses on optimizing other parts of the model. YOLOv4 integrates CSPNet with Darknet53 for a new CSP-Darknet53 backbone, adding spatial pyramid pooling (SPP), a path aggregation network (PAN), and a modified spatial attention module (SAM). The SPP module [20], based on spatial pyramid matching [27], takes feature maps from a convolutional layer as an input and applies a pooling operator at various spatial sizes. These pooled feature maps are concatenated and used as input for the later layers in the model. This operation has been demonstrated to enhance the precision of CNN models [20].

Changes are made to the training process by introducing new data augmentation techniques. Data augmentation helps increase and diversify the training dataset by performing image transformations. Simple augmentation techniques are used, such as flipping and rotating the image. With YOLOv4, Mosaic and Self-Adversarial Training (SAT) is introduced. Mosaic augmentation concatenates four random images from the dataset into a single new image, which introduces the objects in new contexts that improve the model's performance. SAT augmentation involves two stages: in the first stage, instead of having the neural network modify weights, it modifies the image itself, and in the second stage the network learns this new modification.

2.3.5 EfficientDet

EfficientDet [58] aims to tackle two challenges in the object detection landscape, namely efficient multi-scale feature fusion and model scaling. The proposed solutions to these two challenges introduced bidirectional cross-scale connections and weighted feature fusion

(BiFPN) and joint resolution/depth/width scaling for object detectors. BiFPN took the ideas proposed in [31, 34] and improved upon them. Concisely, these improvements included: (1) the removal of feature layers with minimal contribution to the fusion of features within the network, (2) skip connections from input features (P_4 to P_6) to the bottom-up feature aggregation path (skipping the intermediate top-down layer), and (3) treating each top-down bottom-up path as a single layer within a larger feature network.

The EfficientDet model learns a weighting mechanism which lets the network determine the contribution that an input (feature, channel, or pixel) has on the final output feature map(s). EfficientDet introduced this approach due to the fact that certain features at specific resolutions can contribute more discernible information than others with respect to the output feature map(s).

Compound scaling is applied to the backbone, neck (BiFPN), and head (classification and regression) networks. The depth (number of layers), width (number of channels), and resolution (size of input image) are scaled according to a predetermined coefficient, ϕ. The backbone network is unchanged from [56], that is EfficientNet-B0 through B6 is used for $\phi \in \{0, 1, \ldots, 6\}$. The depth of the BiFPN network is determined by $3 + \phi$, the width of each layer within the BiFPN network was chosen to be $1.35^{\phi} \cdot 64$, where 1.35 was chosen via grid search as an optimal scaling factor. Finally, the input image resolution is scaled according to $512 + \phi \cdot 128$.

2.3.6 RetinaNet

RetinaNet [32] (published in 2017) is a one-stage fully convolutional network. RetinaNet has a relatively simplistic architecture, using a ResNet-X-FPN[3] backbone and a parallel classification and box regression head. The classification and box regression heads are both small fully convolutional networks (See Fig. 2.9) responsible for predicting the

[3]The RetinaNet paper, Focal Loss for Dense Object Detection, doesn't claim a specific depth for the ResNet model used in conjunction with the feature pyramid network. However, common implementations use ResNet-50 and ResNet-101.

probability of the type of object and bounding box location at each spatial position. The regression of the bounding box location uses the concept of anchors, which are predetermined (outside of the forward pass of the network) bounding boxes at various locations with varying sizes. The aforementioned spatial positions are merely the features at each level of the feature pyramid network[4]. However, while the model itself was not novel and relatively simplistic, the loss function introduced was innovative. This loss function, coined Focal Loss, aimed to remedy the foreground and background class imbalance problem [42]. The Focal loss function $FL(p_t) = -(1-p_t)^{\gamma}\log(p_t)$ assigned more weight to hard examples (negative examples the classifier fails) ; where p_t represents the estimated probability of the one-hot ground truth label. When $\gamma = 0$, the loss function degrades to generic cross-entropy loss; note too that as $p_t \to 0$ the loss function converges to generic cross-entropy loss and when $p_t = 0$ again degrades to generic cross-entropy loss.

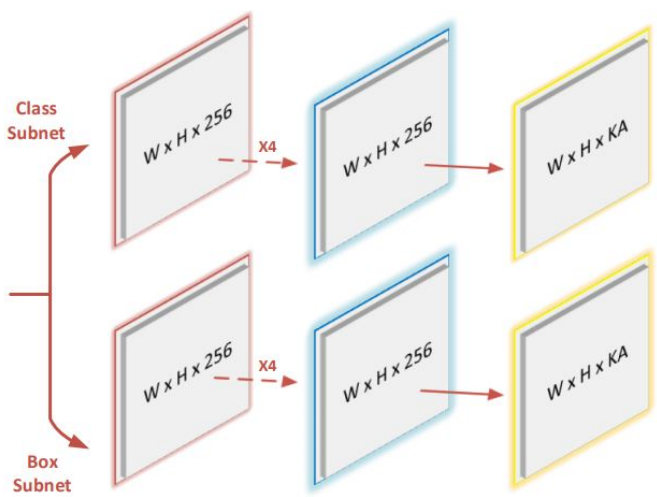


Figure 2.9: Classification and bounding box regression FCN heads used in RetinaNet, reproduced from [32].

[4]Refer to Page 8 of [52] for a more detailed description of what spatial position means in this context.

2.4 Object Tracking

Fruit counting can be performed on an image after detection models recognize the fruit. However, there's a major challenge that must be addressed first. As we input our data in video format, we process around 30 frames per second. If we simply take the count from every frame, that number is completely off the mark due to significant duplication. This is because every frame is treated separately as a unique image, even though the difference between two consecutive frames might be a slight shift due to a camera movement. Fruits appearing in the first frame could be the exact same ones in the next frame. However, if each frame is uniquely counted, then the same fruits are counted twice (and several more times depending on how long they've existed throughout the frames). The solution to this problem is to perform object association across multiple frames, so we can identify individual objects that recur throughout the frames, whilst also detecting new objects, and discarding objects that have left the scene. This would allow us to count the identified objects only once. This is known as object tracking, which is another interesting and popular challenge in computer vision.

Object tracking is the process of associating identified objects in consecutive frames . This is done through saving and comparing objects between frames based on visual appearance and other features. This process has three possible outcomes: (1) the new object is similar to a saved object and is successfully associated, (2) the new object does not match any of the saved objects and is then identified as a new object that had just entered the scene, (3) a saved object hasn't been associated for a while and is deemed to have left the scene and is discarded. There are numerous ways to perform tracking, but they all boil down to matching between frames based on motion or appearance. In the following, we review some of these techniques.

2.4.1 Optical Flow

Optical flow [11] is a popular object tracking solution that instead of relying on features extracted from the objects to be tracked, the brightness of the object throughout its different locations across frames is tracked. The object is assumed to have a small amount of motion, and pixels neighboring the object have similar motions. By taking into account that the brightness of the object remains consistent throughout a video, the object can be tracked by estimating its future locations based on the assumed motion. This concept can be formulated as follows: $I(x, y, t) = I(x + u, y + v, t + 1)$, where the object at location x and y at time t is the same as the object at time $t + 1$ after moving at a velocity of u and v.

2.4.2 Meanshift

Meanshift algorithm [8] heavily relies on the colors of an object in order to track it across frames. Within a set of data points in a "neighborhood" (i.e., a cluster of data that are closely related to one another), the average is computed. Afterwards, the algorithm iteratively shifts data points within the neighborhood towards that average. This is similar in concept to clustering algorithms in machine learning.

In computer vision, colors can be used as data points for the meanshift algorithm. Similar to other tracking algorithms, a region of interest (RoI) is extracted from the frame. This region should contain the target object for tracking and is considered the neighborhood of data points. A color histogram is extracted from the neighborhood. When iterating over the input frames, the extracted histogram is projected onto the frame and the likelihood that each pixel in the RoI belongs to the original image being tracked is computed. If the probability is high, then the RoI shifts towards the new location in which the object exists throughout the following frames.

2.4.3 Deep Regression Networks

Deep Regression Networks [23] is a proposed architecture that's composed of a sequence of two parallel stages of convolutional layers followed by fully connected layers. The philosophy behind the design is to allow the model to analyze two frames, the current frame and the previous frame in which the object was detected. The model learns to search for the original object within the current frame by using the convolutional features. If the object is found, its location is regressed through the fully connected layer. To achieve this, the model must be trained on a substantial dataset that includes images with similar objects in different motions. Through this training process, the model learns to identify the same object through different positions within the frame, angles and poses. This seems like a simple solution and it does show a lot of promise, however it's entirely reliant on the dataset and its training, which may not be practical in numerous situations.

2.4.4 Recurrent YOLO

Recurrent YOLO [67] proposes a fully deep learning based solution. The authors use YOLO for object detection, and to leverage the visual information to identify both features and spatial information about the object (i.e., its location within a scene). Afterwards, this information is then inserted into an Long Short Term Memory (LSTM) network. LSTMs are an extension of recurrent neural networks (RNN). The RNN model is a type of neural network that consists of input, hidden and output layers, and when processing the output, the hidden layer is concatenated with the output from the previous step. LSTM extends RNN by allowing information from all previous steps to get passed along it can carry over to future frames and keep track of temporal information about the object as well. The authors present that LSTM can be used to regress the location of the object, similar to how optical flow predicts future positions for tracking purposes.

2.4.5 SORT

Simple Online Real-time Tracking (SORT) [2] emerged as one of the top solutions for multi-object tracking. The algorithm provides a straight-forward framework that's composed of object detection, motion estimation and data association in order to tackle the tracking problem as efficiently as possible.

During a video feed, frames are inserted into the algorithm one at a time. The first step is to perform object detection on the frames and the algorithm does so by using deep learning-based object detectors (the original paper uses Faster R-CNN). The object detector outputs predicted bounding boxes for objects of interest. Afterwards, initial detections are initialized as new tracks. These tracks save the bounding boxes of the object, as well as some properties that are used for the next step, which is the Kalman Filter [4].

In order to keep track of objects, the algorithm focuses on using motion prediction to guess where the object should be in future frames, then compares new detections against saved tracks to perform data association. To expand, motion prediction is done using Kalman Filter on the saved tracks. The Kalman Filter is formulated as follows: following a constant velocity model, each track has an 8-dimensional state space (x, y, a, h, $\ddot{x}$, $\ddot{y}$, $\ddot{a}$, $\ddot{h}$) which respectively denote the center x and y coordinates of the bounding box, the aspect ratio and height, and their velocities. When new detections are inserted into the algorithm, their bounding boxes are compared against the predicted bounding boxes after motion estimation of the saved tracks. If the two boxes have an intersection greater than a specified threshold, the algorithm declares that the new detection belongs to the track it was compared to and the kalman filter updates its parameters to improve its future predictions. If not, then the new detections are declared as new tracks, and undergo the same process.

2.5 Related Work

Crop yield estimation relies on using image processing algorithms to detect the crop from imagery and count the number of detections to give the estimate. In earlier works, detections are made by distinguishing between the crop's textures and background textures. Wang [62] use visual cues to segment apples from images, specifically the hue, saturation and intensity to detect both red and green apples under specific lighting. Pothen [44] present a keypoint detection algorithm that detects potential fruit regions based on intensity profile then use high-dimensional features in those detected regions to classify them as fruit or not fruit. Roy [50] estimate apple count from images as well as their diameters by segmenting apple clusters in an input image using a nonlinear optimization method. Then, feed the segmented images into a Structure from Motion (SfM) pipeline to reconstruct them up to scale. Malik [40] integrate AI in their approach to yield estimation on citrus fruit. They use k-means clustering in order to perform image segmentation on the fruit, separating the fruit class from the background class. They also incorporate an object separation technique to account for overlapping fruits by computing the neighbor variance in each of the color channels (RGB), then turning it into gray scale and thresholding. This way if there's a color difference among the edges, it will be detected. The aforementioned techniques do not produce state-of-the-art results, are very specific to the selected crop (e.g., apples), and cannot be easily translated into yield estimation for other types of crops without a drastic change in the algorithm. Additionally, these algorithms can perform poorly outside of controlled environments which can hinder color based algorithms, such as light intensity. In our work, the proposed framework can be used with any fruit, given the appropriate dataset is provided to train the model in different circumstances, such as poor lighting conditions and occlusion behind another fruit or leaves.

Deep learning is on the rise in precision agriculture with numerous state-of-the-art object detection and image segmentation algorithms utilizing deep learning models showing

promising accuracy. Furthermore, one of the main advantages of deep learning is generalization, if the model works on one type of object, it must work on other objects. It only needs to be retrained with a suitable dataset relevant to the domain of application. Rahnemoonfar [45] prove this by performing fruit detection on 7 different fruits using the Faster R-CNN object detector which is trained on the respective fruit. Bargoti [1] use Multi-Layered Perceptrons and Convolutional Neural Networks (CNNs) to perform image segmentation to detect and count fruits from images. Sa [51] perform tomato counting using a CNN model that's a modification of Inception-ResNet [55]. Notably, the data they used for training is simulated by generating synthetic images of tomatoes, producing favorable test accuracy on real images of tomatoes. Chen [7] present a deep learning pipeline for counting apples and oranges. An image of the crop is input into a blob detection neural network that is a fully convolutional network [39] which outputs a segmented image to distinguish fruit pixels from non-fruit pixels, noting that there could be a cluster of fruits that's referred to as a blob. They then use a count neural network, which is another CNN that takes bounding boxes around each detection and estimates the count of fruit in that box. Koirala [26] modified the YOLO architecture, referring to it as MangoYOLO, to detect mango fruits in images of tree canopies, proving YOLO's capability in detecting mangoes. Bresilla [3] use a single shot detector architecture to detect apples and pears. Zhao [69] use Mask R-CNN for strawberry detection. These works produce great accuracy ranging between 90% to 95% with good performance and potential for generalization, showing that deep learning is a viable methodology for yield estimation. However, they all work on static images, not videos, of crops. This is a major limitation in these approaches as footage of crops are typically captured in a video format. In our work, we use the detection model within a tracking pipeline, which allows the system to keep track of detections across video frames and count new detections only once.

Liu [38] address counting through videos by using a fully convolutional neural network

(FCN) for object detection and optical flow using Kanade-Lucas-Tomasi (KLT) tracker. However, their tests on apples were run under controlled illumination. Moreover, KLT-based optical flow tracking has shortcomings: (1) it relies on the brightness of different regions in the frames remaining consistent, (2) the object motion in that region has to be consistent. Any change of lighting or sudden change of motion will cause the tracker to suffer. Their work also included the use of SfM to avoid counting apples in different tree rows, which works well but the algorithm itself is very computationally intensive. In our approach, we use a single-shot-detector (SSD) which is considered the state-of-the-art solution to object detection. Additionally, we developed a custom tracking pipeline that leverages deep learning and intersection over union (IoU) matching in order to properly track objects through sudden motions and changes in appearances. We also design a lightweight solution to resolve apples in back rows being unintentionally detected. Our dataset and tests are run on apple trees in natural illumination during daytime.

One of the branches in precision agriculture is the development of a decision support system for farmers. Giustsi [15] developed a fuzzy logic system to support farmers in irrigation decisions. Their model analyzed the level of moisture within the soil, and they set mechanisms to maintain safe levels of irrigation. Zhang [68] also introduce a system to aid in monitoring and managing irrigation via the integration of GIS technologies. The use of geospatial information is substantially useful when developing a decision support system and when presenting the data mapped and visualized to the farmer. Reddy [46] discuss the use of GIS for crop management, national and regional policy, and other aspects of farming. They conclude that the use of GIS plays an important rule in a stronger understanding of the analysis performed by various sensors. In our work, we introduce a smart harvesting system. To our knowledge, none of the popular works on yield estimation discuss the use of spatial data when presenting the yield. While yield is being counted in our pipeline, we also synchronize GPS coordinates with counts taken at their respective location. This was made possible as we had a GPS recorder running

during the capturing of fruit footage. We then introduce a container placement algorithm that uses the combined GPS and counts to optimally assign containers locations within the field before harvesting. Through our proposed system, not only do we provide the farmer with a breakdown of their yield, we also aid them in efficient management of resources.

2.6 Summary

In this chapter we introduced the use of deep learning within the context of computer vision. The use of deep learning in computer vision has been the standard since the conception of Convolutional Neural Networks (CNNs). Convolutional Neural Networks are the de facto standards for object detection. We discussed various object detection models that utilize CNNs. This chapter also presented numerous related yield estimation frameworks and outlined research gaps and research objectives.

Chapter 3

Yield Estimation from Video Feeds

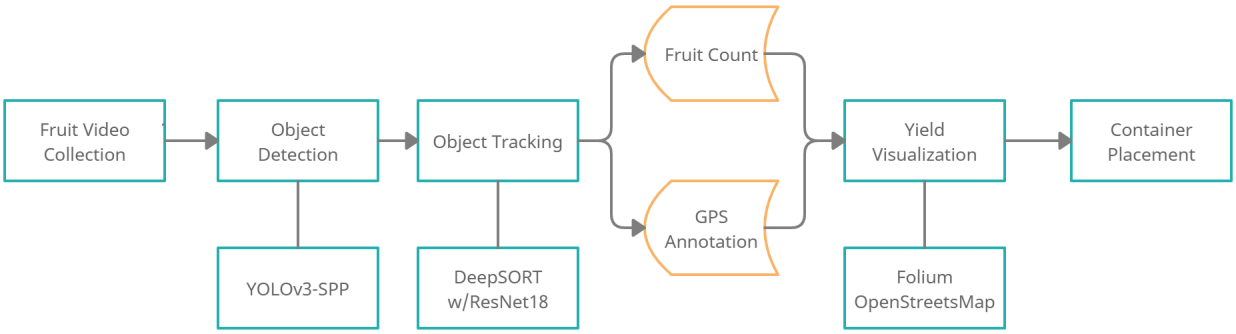


Figure 3.1: An end-to-end overview of the proposed smart harvesting pipeline

3.1 Framework Overview

We provide a high-level view of the proposed yield estimation and smart harvesting assistance framework in Fig. 3.1. We begin by inserting video frames into the processing pipeline. Object detection is applied on each frame to detect the fruits within the frame, however as discussed earlier, detecting the fruits is not enough to count the fruits in a video. The detections are passed onto our modified DeepSort tracking model. The tracker is responsible for creating associations between objects on the consecutive frames. For example, if fruit "a" is present in the first and second frames, then the tracker associates between them and gives them an ID of 1. Once an individual fruit is identified and tracked across the frames, it is counted, and a total yield estimate is provided at the end. During the counting process we also perform geospatial annotation, leveraging the captured GPS coordinates in the videos. We annotate every GPS point with a count based on the GPS device frequency (e.g., one GPS coordinate every one second and the count within that one second are recorded together). We develop an algorithm that synchronizes between geospatial data and yield estimates, which outputs a file with GPS points and counts at those points. We then visualize this information on a map, so the yield and its location are presented in a detailed and presentable manner to the farmer. We then formulate an optimization problem to place containers used for harvesting, such

as apple bushels, on the map based on the recorded geospatial information. This smart harvesting solution extends the usability of yield information and informs the farmer of the exact logistics needed for their harvest. In this chapter, we explain each step in our approach to providing a fully automated approach to yield estimation and visualization, and how we provide optimal container placements for smart harvesting.

3.2 Fruit Detection

Fruit detection is the first step in the proposed pipeline. This is primarily an object detection task, where the desired objects (in our case, the fruits we are counting) are detected using bounding boxes. These boxes contain pixel-wise coordinates and surround each detected fruit in a scene. The model also adds the classification of the fruit, as well as the confidence score. Due to the prevalence of existing powerful general use object detection models, we explore the possible options and make a decision based on accuracy, performance, robustness and ease of integration with our tracker.

Model Selection

Convolutional neural network (CNN)-based object detectors have been able to achieve state-of-the-art performance on various benchmark datasets. As discussed in Chapter 2, the basic structure of a CNN object detector contains two parts: the backbone [12, 57, 61, 5, 41, 65] which is responsible for extracting features from the image(s), and the head [48, 58, 33, 71] which is responsible to predict bounding box locations and class values. Recently, CNNs have been fit with intermediate blocks between the backbone and the head; commonly called the neck [21, 30, 36, 64, 6, 35] which is responsible for collecting feature maps from multiple stages of feature extraction or feature refinement for feature layers. Commonly, the goal of the intermediate neck layer/blocks is to improve accuracy whilst minimizing the incurred overhead. In Chapter 2, we explored

various object detectors, each with its own unique architecture and set of advantages and disadvantages. With single stage detection (SSD) models showing high accuracy without processing speed trade-offs, we discard multi-stage detection models as they show significantly slower performances in comparison. We compile reported performance of the best object detector candidates in Table 3.1 and analyze their performance in terms of Average Precision (AP) and Framerate Per Second (FPS).

We chose to use YOLOv3 with spatial pyramid pooling (YOLOv3-SPP) [48] [25] as the back-end object detector for our pipeline. Table 3.1 shows that YOLOv3-SPP has the best trade-off for accuracy and speed compared to other current state-of-the-art object detectors. For example, EfficientDet-D3 shows the highest AP of 47.2%, however the FPS performance is notably lower than both YOLOv3 models. We observe that RetinaNet boasts the highest FPS with AP that rivals YOLOv3, however the results are somewhat skewed by the use of NVIDIA TensorRT (TRT). With the use of TensorRTthe network's inference time could potentially decrease without affecting the accuracy. However, we do not use TensorRT in our work because it's still highly experimental. The framework is very hardware demanding as it only works on certain NVIDIA GPUs and is only compatible with limited CUDA versions. More importantly, YOLOv3 is a heavily studied object detector that's used in many research and industry applications- many bugs and potential issues are ironed out in comparison to fresher models.

YOLO3 Model Architecture

You Only Look Once (YOLO) object detector is divided into two components: feature extractor (the backbone) and detector (the head). The feature extractor is responsible for getting a feature map of the image. YOLO uses a CNN model called DarkNet53 that uses 53 layers as shown in Fig. 3.2. The architecture is fairly straightforward with multiple convolutional layers to process the input, and residual layers, which are also known as skip layers, in-between. Firstly, convolutional layers use filters to analyze

Table 3.1: Comparison between different SSD models

Method	Backbone	AP	FPS (Batch Size of 1)	GPU
YOLOv3-512 [48]	Darknet-53 [47, 48]	42.4%*	48.7*	Telsa P100 (10 TFLOPS)
YOLOv3-608 [48]	Darknet-53 [47, 48]	43%*	43.1*	Telsa P100 (10 TFLOPS)
EfficientDet-D3 [58]	EfficientNet-B3 [57]	47.2%	34.4	Tesla V100 (15.7 TFLOPS)
RetinaNet [33] (w/TRT)	SpineNet-49S-640 [12]	39.9%	85.4	Tesla V100 (∼30 TFLOPS)
RetinaNet [33] (w/TRT)	SpineNet-49-640 [12]	42.8%	65.3	Tesla V100 (∼30 TFLOPS)
CenterNet-HG [71]	Hourglass-104 [41]	45.1%	7.8	Titan XP (12 TFLOPS)
CenterNet-DLA [71]	DLA-34 [65]	41.6%	28	Titan XP (12 TFLOPS)

*Achieved via custom tests through evaluating the model based on the default COCO benchmarks.

blocks of an input matrix, starting with the initial input image. Filters are considered the weights within the context of CNNs. They are small two-dimension matrices that are multiplied with each block of the input matrix. After all blocks of the input are used in the computations, the output is a new matrix that contains feature information. This process can be repeated multiple times by stacking subsequent convolutional layers. The output of the convolutional layer can be considered a mathematical representation of the image that's used to classify the object within the image. Theoretically, the deeper the network is, the more accurate the classification will be. However, that's not always the case, hence the introduction of residual layers [22] which are also used in Darknet53. In traditional neural networks, where each layer feeds directly into the next layer, residual networks allow for some layers to be skipped. This combats the vanishing gradients problem, where changes in the weight parameter (gradients) become so small that they disappear and that information becomes lost. Thus, during back-propagation no useful information will be sent back, crippling the training process. With skip connections, these small gradients can back-propagate through shorter paths, allowing for less changes and that information becomes preserved. The outputs of convolutional layers are considered the feature maps, which are used in the next stage.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.2: Darknet53 architecture that is used as YOLOv3's backbone [48]

Detectors in the YOLOv3 leverage the output of each of the last 3 convolution stages. Three stages are selected to allow for multi-scale detection, where smaller objects are easier to detect in earlier stages, and bigger objects are easier to detect in later stages. In the YOLO model we have integrated, the spatial pyramid pooling (SPP) approach is used. SPP is based on the SPM model [27]. It focuses on improving classification and detection on a multi-scale level. SPP utilizes the output feature maps of different stages within the DarkNet CNN (specifically, the outputs sized $32x32$, $16x16$ and $8x8$). After the feature maps are extracted, max pool operation is performed. The outputs are then concatenated together to form a long single vector similar to Fig. 3.3. By using multi-scale features, the model is able to gather spatial information that significantly improves scene interpretation and detection of objects in various aspect ratios and scales. In practical settings, different fruits have different sizes (e.g., apples are smaller than pumpkins), plus the perspective of the footage can affect the fruit size as well (e.g., aerial views of pumpkins lead to pumpkins appearing smaller in size). As such, it's vital that

our detector has the versatility to function accurately with any object size. Additionally, the size of the vector is fixed length, combined with the fixed scales of the design, it allows the model to work on images of any size.

In YOLO, the input image is divided into a 2-dimensional grid containing NxN cells (N being the grid size). Each cell within the grid is analyzed simultaneously through the feature extraction and detection explained above. Following the anchor box detection method, anchor boxes are assigned to each cell in the output matrix of the convolution stages. These anchor boxes have different aspect ratios and sizes. During the learning phase, the anchor box with the best intersection with the ground truth bounding box is considered the correct prediction. Thus the information predicted for each box is the location of the box (pixel wise center coordinates of the box denoted by x and y), whether the box contains an object or not, and what class this object belongs to.

Non-max suppression (NMS) is applied after all boxes have been predicted. This is important because it's possible for the same object to be detected multiple times. The algorithm takes the box with the highest confidence, then compares it with all other potential boxes. If the intersection over Union (IoU), seen in Fig. 3.4, is above a certain threshold (for example, an IoU threshold of 0.5 means that if more than half of two objects intersect), then it is removed as it will be considered a redundant box that covers the same object. This step is performed on all predicted boxes. It's important to allow for some intersection to exist in apple detection, as having multiple apples overlapping occurs often in apple trees.

For our work, we use YOLOv3 with the default configurations to detect apples, where we obtain the x, y coordinates of the center of the box; the width and height of the box; the class it belongs to; and the class confidence score. All this information is then used for tracking. The use of default configurations allows us to improve the accuracy of detection directly using transfer learning, where we can simply fine tune the original weights pretrained on the MSCOCO dataset which already contains apple data. We

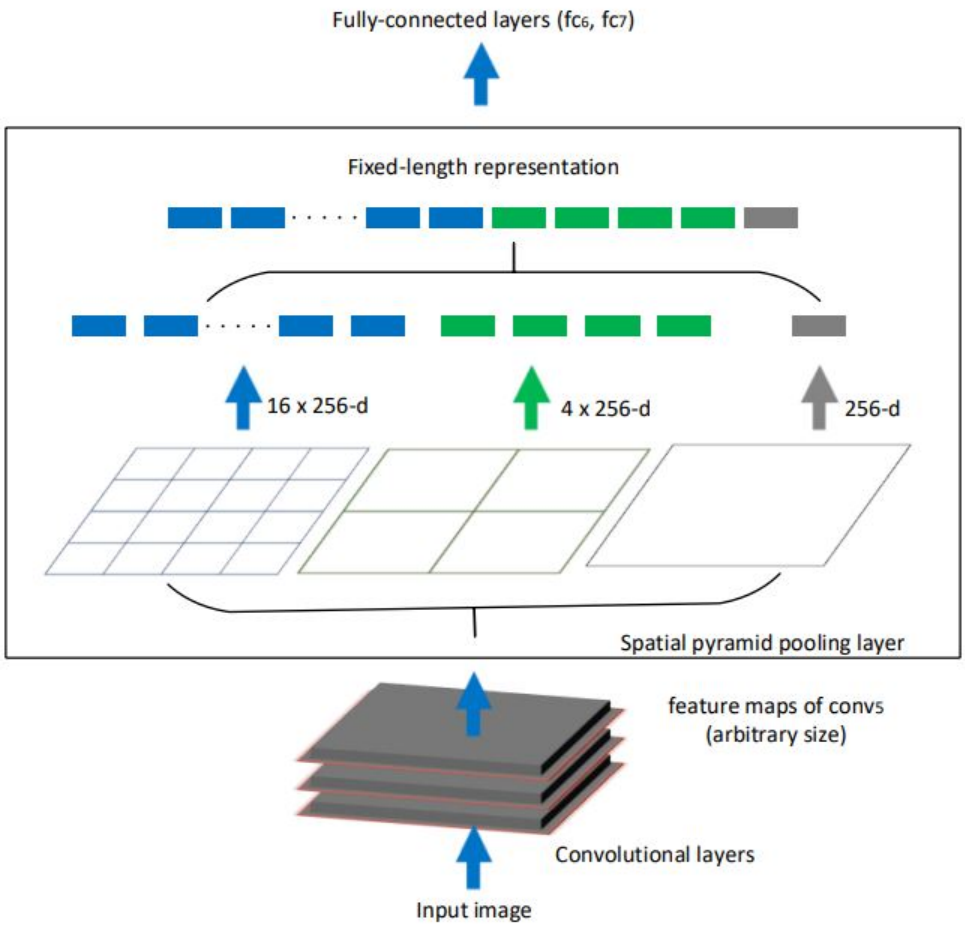


Figure 3.3: Visualization of SPP, reproduced from [20]

provide more details about dataset preparation and training in Chapter 4.

Correcting Apple Detections

In our implementation with apple data, we noticed that apples from other tree rows are being detected. This is not intended as apple orchards are divided into rows of trees, and we perform the counting task on one row at a time. We created a lightweight thresholding approach to adjust the detections returned by YOLO to overcome this problem. We ran a standalone test of the detection model, analyzing the average bounding box sizes of the distant apples. Based on our observations, we adjusted the bounding box height and width thresholds to 30 pixels after running several experiments with varying thresholds.

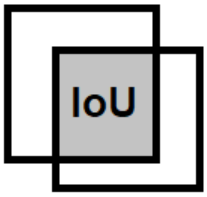


Figure 3.4: IoU denotes how much two boxes overlap with each other, with 0.0 being no overlap, and 1.0 meaning they fully intersect

Any bounding box that is smaller than this threshold is eliminated.

3.3 Fruit Tracking

We extract the bounding boxes from the detector, and perform a tracking function to keep track of the objects across the frames. This type of tracking function is considered a multi-object object tracking, or Multi-Hypothesis-Tracking (MHT) [28], where there are numerous objects in the frames that are tracked separately. The algorithm we use for MHT is a modified version of DeepSORT [63].

DeepSORT is an extension of the Simple Online Realtime Tracking (SORT) [2] algorithm. It relies on creating " tracks" which represent the tracked objects and applying Kalman Filter to predict the next states of objects. DeepSort then associates between the same objects in different frames by using distance metrics, one is based on the motion of the object and the other is based on the appearance of the object. The model gives favorable performance only using the appearance as a metric [63], so for our work, we only use the features produced by the CNN which represent the appearance of the object.

Following the original SORT, DeepSORT performs Kalman Filtering and defines the tracking scenario on an 8-dimensional state space (x, y, a, h, $\ddot{x}$, $\ddot{y}$, $\ddot{a}$, $\ddot{h}$), which respectively denote the center x and y coordinates of the bounding box, the aspect ratio and height, and their velocities. A linear Kalman Filter following a constant velocity model is used, which is a standard for object tracking [29]. The bounding box coordinates (x, y, a, h)

which are received from YOLO are used without modification. DeepSORT then uses deep learning and CNN features to create associations between new detections and the current tracked objects by using a minimum cost algorithm to obtain the objects with the minimum distance with their associated tracks. In the following sections, we describe the components of the tracker in detail.

Track Initialization

After we obtain the first bounding boxes from YOLO, the first step in the DeepSORT pipeline is initializing the tracked object for the detections. The track is given a number of attributes including:

- Track ID as an identifier
- Age which denotes how many frames this track existed in, initialized as 1
- Number of hits which is incremented every time this track is successfully associated, initialized as 1
- Feature matrix which is the output of the CNN and represents the object's appearance
- Track state which represents the current state of the track so it can be confirmed or deleted, initialized as tentative
- Mean and covariance matrices which are used for the Kalman Filter that are updated with every prediction

Convolutional Neural Network

DeepSORT's main feature is the use of deep convolutional features. When an image is input into a CNN, the features of the image are analyzed within each layer and then an activation function such as Softmax is applied to the output vector to classify the

object once all the convolutional layers have been completed. Intuitively, this means that the vector output after convolutional layers describes the appearance of whatever is in the image. This was leveraged in YOLO in order to detect objects and it's also used in DeepSORT's pipeline as the appearance metric for tracking. The use of appearance allows for better tracking in case of a random object motion and occlusion, which are both very common challenges in tracking applications.

The original model used is a wide residual network that's been trained on a person re-identification dataset [70] which is limited to people tracking (specifically, pedestrians) making it unsuitable for any fruit data. Thus, in this work we insert a different CNN model that shows a state-of-the-art performance called ResNet18 [22]. We chose to use ResNet rather than creating a CNN model from scratch for several reasons. First, ResNet has been utilized in a wide range of applications for years and its performance and limitations have been well investigated. Second, ResNet comes with a variety of setups, primarily varying depths. This allows us to choose a depth based on our requirements. We chose ResNet18 for our system because it has the best performance and fruit characteristics aren't complicated enough to justify more layers. We also performed some initial tests and found no difference in counting accuracy between ResNet18 and ResNet101. Fourth and most importantly, ResNet provides weights that are pretrained on the ImageNet classification dataset [10], a massive dataset with 1000 classes, including fruits such as apples for our application. This eliminates the requirement to train the DeepSORT feature extractor and allows the DeepSORT pipeline to be used for practically any fruit without modifying the tracking model and saving time and resources that would otherwise be spent on training.

Fig. 3.5 shows the architecture of ResNet18. There are 5 convolutional layers, and a skip connection is in between every layer. After the final convolutional layer, there is an average pooling layer which leads to an output vector of size [1, 1, 512] that's input into a fully connected layer for classification. Since we do not perform classification and we

need the appearance descriptor of the object, we take the output of the average pooling layer as our final output to be treated as the feature map of the object.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Figure 3.5: The ResNet18 [22] architecture is used for DeepSORT's feature extraction. The fully connect and Softmax layers are discarded

Association and Counting

The model associates between tracks and detections in a frame using their respective feature maps. This is an assignment problem to be solved using the Hungarian algorithm, which is a standard minimum cost algorithm. It's vital to use an efficient algorithm as the number of tracks and detections can increase substantially, where every track needs to be compared against every detection. The problem shown in Eq. 3.1 is formulated as follows: given an array of tracks (T) of size t, and an array of detections (D) of size d, we compute a new cost matrix (C) where index [i, j] is the cosine similarity between the feature map of T(i) and the feature map of D(j). After the matrix is complete, we simply find the minimum cost for each track in T out of all the detections and that forms the association.

$$C(i, j) = \min(1 - cos\_similarity(D(j), T(i))) \tag{3.1}$$

The Hungarian algorithm can be divided into 4 main steps. We start off by providing our input, which is a 2-dimensional matrix sized n x n, each cell representing the cost between the track and the detection. The first step is to obtain the minimum value in each row and subtract it from each cell in its respective row. The second step is to do the same for each column, i.e., subtracting the minimum value in each column from all elements in that column. The third step is to mask rows and columns to cover all the zeros that were computed due to the previous two steps. Then there are two scenarios: (1) the number of lines required to cover zeroes is n, and in this case the optimal assignment is done and the algorithm ends, (2) otherwise, we search for the smallest uncovered index and subtract it from all uncovered indices. If the entire matrix is covered, then the algorithm ends, otherwise repeat this step until completion. The final optimal assignments are the cells with the value '0'. Meaning, if $C(1,3) = 0$, where 1 is the index of Track 1, and 3 is the index of Detection 3, then Detection 3 will be associated with Track 1.

The cost value in our context is the cosine distance between CNN features. After a feature map is extracted from the ResNet18 model we implemented, we obtain the cosine distance using Eq. 3.2. The cosine distance, also known as cosine similarity, is a method to compute how similar two vectors are in an inner product space. Essentially, the cosine similarity measures the angle between two vectors and applies the cosine function to it. This determines where both vectors are pointing and whether they're pointing in the same direction. This is applicable with CNNs as we can obtain feature maps in the form of a two-dimensional matrix from the convolution and pooling operations that we can then flatten into vectors and compare between them. For example, Fig. 3.6 shows the saved image which we can assume as track 1. If there are two new detections added to our tracker, shown in Fig. 3.7 and Fig. 3.8, we now compute the cosine similarity according to Eq 3.2. The result will look something like: $Similarity(i,j) = [0.64, 0.99]$. That means the similarity between Track 1, Detection 1 is 0.64, and the similarity between Track 1, Detection 2 is 0.99. Then, once a minimum cost algorithm is performed based

Figure 3.6: First apple detected in first frame and is inserted into the tracker which identifies it as track 1.

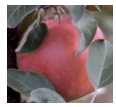


Figure 3.7: Apple identified in second the frame, noted as detection 1 is tested for association.

on Eq. 3.1, Track 1 will be associated with Detection 2, which is the correct assignment as they are quite clearly the same apple with only a slight shift in motion. This operation can be extended to any tracks and detections that will be added.

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{3.2}$$

As the objects get analyzed and tracked throughout the frames, there are three states that could be assigned to each track in every frame:

1. Tentative: this is a temporary state. It means that a new detection is potentially a new object to be tracked. New detections remain tentative until they are either matched with an existing track or are turned into their own new track.

2. Confirmed: this means that the track is created and is confirmed as a new object to have entered the scene. The detection changes its state from *Tentative* to *Confirmed* and is ready to have new detections associated with it.

3. Deleted: this means that the track has left the scene and is no longer tracked. It

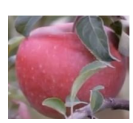


Figure 3.8: Second apple identified in the second frame, noted as detection 2 is tested for association.

is considered deleted and will no longer be considered during the matching stage.

Therefore, a track is not immediately confirmed upon association. The track needs to have a number of hits greater than 2 to change its state to *"Confirmed"*. This helps to avoid brief false detections. Secondly, not all tracks will be associated. The distance (where 0.0 denotes an exact match, and 1.0 denotes completely different features) needs to meet a certain threshold to be considered as a match. Following the original implementation, the maximum distance is 0.15, meaning that for anything higher, the match will be discarded. If a track is not associated, it will remain saved for a number of frames until its age surpasses the preset maximum age, which we set as 30. In a 30 FPS video, if an object is not associated for 1 second, it is discarded as it is considered to have left the scene. Before unmatched detections are initialized as new tracks, they undergo IoU matching first. Specifically, the IoU between two tracks bounding boxes is computed, and if the IoU is 0.3 or greater (meaning that there is some intersection), the tracks are associated with each other. This improves the tracking robustness because sometimes a fruit can be clear in one frame, obscured in another (thus, having different appearance features), then clear again in the following frame. The IoU tracking helps keeping track of that apple when an appearance association can briefly fail. Another important parameter in DeepSORT includes a minimum confidence, which is the minimum accepted confidence from the detector. Anything lower than the minimum confidence causes the detection not to be inserted into the tracking pipeline. We selected a score of 0.4, as fruits that were partially hidden by leaves or branches, or fruits that were blurred due to camera motion were indeed detected, but with notably lower confidence. Thus, a lower threshold allows for such apples to be considered. It was also observed that lower thresholds than 0.4 include wrong detections, mostly leaves that were mistakenly detected as another fruit and have very low confidence scores.

It's important to note that, while every new track has its own ID, we can't use it as a count reference because some tracks are still tentative and will be deleted if they aren't

properly associated. As a result, apples are counted only when the track is confirmed.

3.4 Geospatial Mapping of Fruit Count

The combination of geospatial information with crop analysis is critical to smart harvesting as it provides farmers with rich information to optimize their resources and make informed decisions on how to plan for pre- and post-harvesting. GPS technology can enable feature maps in the field such as visualization of soil happiness, tree density, targeted treatment and fertilization, container placement and many more. In this work, we record GPS points while recording videos of the apple rows to support these features. GPS capturing is synchronized with video capturing so we are able to match the GPS coordinates with the video during the counting process.

During data gathering, the GPS device consistently records its current coordinate. After data capturing and analysis are complete, we annotate the counts from the video frames with their respective location on a map. To accomplish this, we record the frequency at which the GPS points are captured and assign a GPS point to a set of frames. In this research, we record 1 GPS point every 3 seconds (GPS period). Our video is recorded at 30 frames per second, which means 1 GPS point is assigned to every 90 frames (3 * 30). We formulate an equation for the frames per GPS point in Eq. 3.3. We then implement a counter alongside our fruit counting pipeline that increments until it reaches the maximum frames per GPS point. As such, the count is precisely recorded with the exact geolocation. The counter is reinitialized after each GPS annotation and the following GPS point is ready to be assigned. This process is presented in Algorithm 1.

$$frames\_per\_GPS\_point = GPS\_period * frames\_per\_second \tag{3.3}$$

The GPS coordinates and their recorded count (shown in Fig. 3.9) are used to

Algorithm 1 GPS to Frame Mapping Algorithm

```
procedure GPS_TO_FRAME(apple_vid, GPS_sheet)   ▷ Input the video and a sheet of recorded GPS
    frames_per_GPS_point = GPS_period * frames_per_second
    while video_frame_id ≠ end_frame_id do
        fruit_detections = YOLO(video_frame_image)
        fruit_count = DeepSORT(fruit_detections)
        temp_count += fruit_count
        if( frame_count = frames_per_GPS_point)        ▷ Period reached
        write(GPS_coordinates, temp_count)             ▷ Record GPS and count
        temp_count = 0                                 ▷ Reset period and current count
        frame_count = 0
    return Updated_GPS_sheet                           ▷ GPS sheet with respective counts
```

provide better visualization of the data. Our aim is to create a map containing the yield information at a point-by-point basis on a map. To do so, we use the Folium package in Python and create a map containing GPS points with a tooltip containing the count in an OpenStreetMap template. A runtime visualization is shown in Fig. 3.10.

1	Lat	Lng	Count
2	43.91709	-78.6277	37
3	43.91709	-78.6277	61
4	43.91709	-78.6277	43
5	43.91709	-78.6277	43
6	43.9171	-78.6277	44
7	43.91711	-78.6277	66
8	43.91712	-78.6277	48
9	43.91714	-78.6277	55
10	43.91716	-78.6277	71

Figure 3.9: A sample from the GPS data in an excel sheet after counting

3.5 Container Placement Optimization

We utilize the geospatial information and yield estimation to provide farmers with a smart harvesting system that will assist them with harvesting logistics. To demonstrate this feature in this work, we perform container placement optimization where we provide a strategy to place a minimum amount of containers in optimal locations across the field when preparing for harvesting.

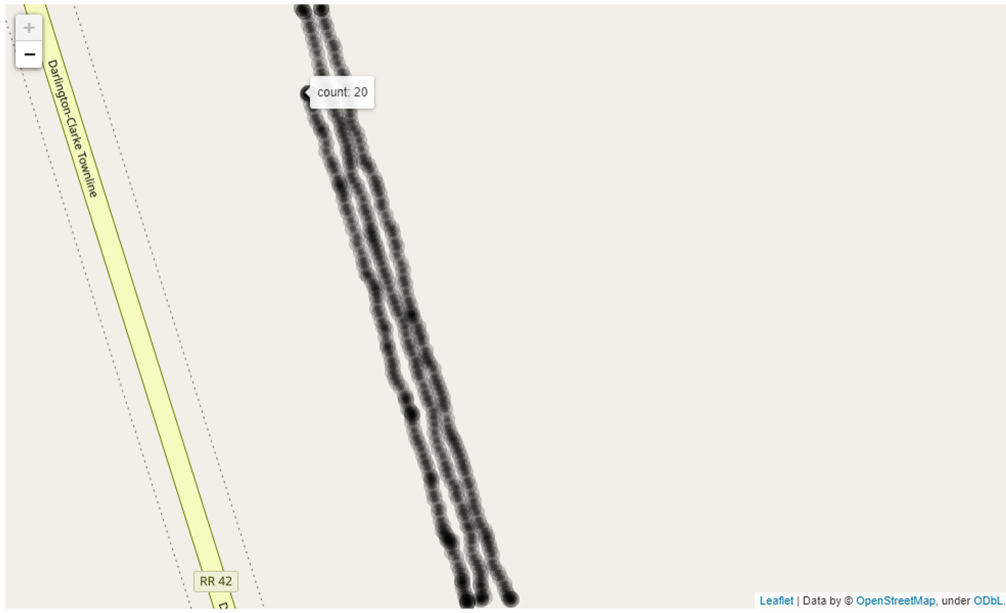


Figure 3.10: Mapping the GPS data after counting

To formulate the objective function for this task, we use the sum of the vector representing the number of possible containers we can use s3.4. To clarify, we have $y_{\mathrm{k}} \in [0, 1]$ for $k = 1, 2, ..., K$, where K is the upper-bound, or maximum number of containers to place. This means that if there's 1 container assigned, it would be a summation of 1+0+0+..., leading to a total of 1. If there are 2 containers assigned, then the summation would look like 1+1+0+..., leading to a total of 2 and so on.

$$\min \sum_{i=1}^{K} y_{\mathrm{k}} \tag{3.4}$$

Our constraints following this are as follows: we have $i = 1, ..., n$ apples, we introduce a variable x_{ik} that is used to determine whether or not apple i is assigned to container k. Note that an apple must be assigned to only one container. To ensure that every apple is assigned to a container only once, we introduce the following constraints:

$$\sum_{i=1}^{K} x_{\mathrm{ik}} = 1 \tag{3.5}$$

What this constraint implies is that every apple "k" is assigned to its respective

container "i". Moreover, "x" is a matrix that contains information about each apple and gives apple "k" the value of 1 at its respective container i, and zero in the rest of the column. For example, in the below matrix, with 3 apples and 3 containers, apple 1 is assigned to container 1, apple 2 is assigned to container 3, and apple 3 is assigned to container 2.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

The last constraint that is needed is the distribution of weight. We check to see that the weight of the apples assigned in every container does not exceed the maximum weight a container can carry. This is done by taking the sum of the product of the total number of apples in a given container "x" with respect to the weight of each apple "w" and determining if it is less than or equal to the maximum weight "m" of a given container "y". This is represented as follows:

$$\sum_{i=1}^{n} w \cdot x_{\mathrm{ik}} \leq m \cdot y_{\mathrm{k}} \tag{3.6}$$

Finally, in our implementation, we monitor the distance between each two containers to ensure they don't exceed a set maximum distance. Once the second container reaches the distance threshold, it's immediately placed even if it's not full. This is to ensure that large containers are not too spread from each other, leading to inconvenience for the pickers during harvesting.

The final output is the optimal number of required containers and their GPS coordinates on a map. The visualization is implemented using the Folium package in Python. By offering this service to farmers, they will be able to plan harvest routes, equipment and container expenses, and labor requirements more efficiently, thus transforming the future of precision agriculture with advanced technologies.

3.6 Summary

This chapter provides an overview of the proposed yield estimation and smart harvesting framework and describes each component in detail. In this framework, YOLOv3 is used as the object detection model due to model efficiency, maturity and stability. A modified version of the DeepSORT tracking algorithm is used to track detected objects across multiple frames to avoid double counting. In the modified version, we implement a different feature extractor with the DeepSORT algorithm to make the model more robust and fruit independent. Additionally, we present our mechanism for annotating GPS coordinates with fruit counts to support efficient container placement and create a smart harvesting solution that can give the farmer clear information about the yield in the farm. The proposed optimal container placement solution leverages the geospatial information and yield data to provide the farmer with recommendations with container placements that can help them plan their storage and harvesting costs.

Chapter 4

Performance Evaluation and Discussions

4.1 Introduction

In this chapter we present our strategy in annotating the fruit datasets used in our experiments. We found that precisely annotating our dataset to include visual challenges, such as various brightness and fruit occlusion, plays a vital role in improving the performance of the detector, and by extension, the rest of the pipeline. We discuss our logic behind our dataset preparation and show our performance results in different cases. We initially test on small videos on apples and test the scalability of our pipeline on significantly larger footage of apple rows. We additionally test the effectiveness of our DeepSORT modification, and the versatility of our pipeline by testing on video feeds of oranges and pumpkins. Lastly, we show our container placement algorithm results on one of the tested apple rows with different constraints, providing optimal container placement locations while showing how much of each container is utilized.

4.2 Dataset Preparation

The massive success of deep learning is generally attributed to its abilities to learn a wide variety of features from data. As such, preparing a thorough dataset is the most important step within a deep learning-based pipeline. In our work, we train the object detector model first on separate small fruit datasets without training the feature extractor in the tracking module. The reason for using small domain-specific custom datasets is that modern CNNs are generally pretrained heavily on massive image datasets, namely ImageNet for feature extraction [10] and COCO for object detection [29]. ImageNet is a massive dataset that consists of millions of annotated images and around 20,000 categories, from humans to various types of fruit. ResNet (our CNN feature extractor) is one of the models that has already been trained on ImageNet, and functions efficiently for fruits. This allows us to save time and effort needed to prepare a fruit dataset just for tracking and allows our tracking module to be used immediately for most fruits. On

the other hand, the COCO dataset is geared towards object detection and segmentation tasks, containing hundreds of thousands of images with annotated objects and around 81 classes. Modern object detectors are pretrained on the COCO dataset, however, as it's relatively on a smaller scale, it's generally required to fine-tune the weights of object detection models to work on the desired objects. While we investigate the accuracy of using pretrained weights in a couple of cases (apples and oranges) that will be shown in the following section, we also prepare three small fruit datasets (2 apple and 1 pumpkin) to train our YOLO model with.

To begin with, we prepare two separate apple datasets using footage taken from an apple orchard containing around 150 100 images, respectively. The first was used for early experiments and was taken at a time when the apples were not fully ripe, and the second one was used for yield estimation of full rows of apples completely ripe. To annotate our images, we use a label software called YOLOLabel. To annotate images for model training, we draw boxes around the desired objects provide a label to it (i.e., the desired class). Each image has a respective text file that contains the annotation details. Specifically, the text file contains the center x and y coordinates of the box, the width and height dimensions, and the class ID. In our work, we load the apple images and specifically annotate the apples. Given that we run our work on apple videos, we configured the model so that we only have 1 class: *"apple"*, so all our objects have a class ID of 0. If we use the YOLO model with pretrained COCO weights, then *"apple"* ID is 48. Fig. 4.1 shows what the UI of the application looks like and an example annotated image.

It is crucial to properly annotate images as such annotations determine what the network learns. The network is efficiently capable of learning simple features, where apples are obvious and not obscured. However, different image quality and context conditions in which apples might not be very clear to the object detector must be considered for efficient training. In apple orchards, it's quite common to find apples in contexts that

affect their visual appearance. For traditional methods, poor lighting or occlusion would pose significant challenges for appearance-based detections, however, with deep learning we can train our network to learn such features. As such, we need to consider the following conditions while annotating the custom dataset: apples that are hidden by leaves, apples that overlap with each-other, apples under different lighting conditions, and other conditions that could potentially affect their visual appearance. Table 4.1 highlights the main different conditions that we've addressed during annotation. By labeling the apples within these different contexts, we significantly improve the model's robustness whilst addressing key challenges (mainly occlusion and brightness) with fruit detection.

Since we aim to develop a general fruit detection approach, we also prepared a small pumpkin dataset that consists of 30 images of pumpkins (an example is shown in Fig. 4.2). The images mostly consist of aerial views of pumpkin patches. Unfortunately, we couldn't acquire more data as pumpkin data is severely limited. We use the same software package for annotation and follow the same labeling philosophy discussed above.

4.3 Fruit Counting Results

For performance evaluation, we compute the accuracy by comparing between the yield predicted by the model and ground truth count which is performed manually in our case.

Table 4.2 shows the result of running our pipeline on the apple videos. We ran the experiments over 3 different model configurations: fine tuned weights, pretrained weights, and pretrained weights corrected by our apple size thresholding mechanism . Table 4.3 shows the results of running our pipeline on full, five minutes long videos of three complete rows of trees. These videos were taken closer to harvest season. In this video, apple trees are well populated with fruits, which is good to test the performance of the proposed pipeline. We used fine tuned weights for all three rows and a correction mechanism was not needed for this experiment as we did not annotate distant apples in our dataset,

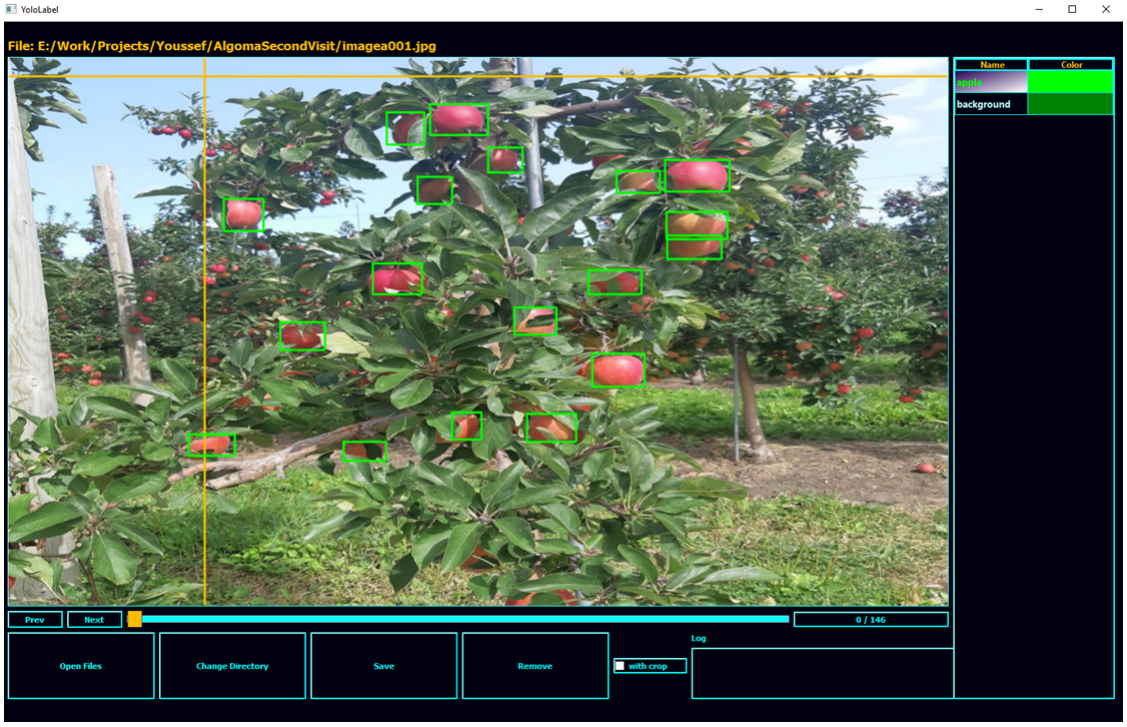
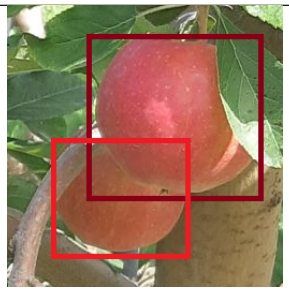


Figure 4.1: YOLOLabel package provides an easy-to-use and efficient annotation tool using bounding boxes

Figure 4.2: A sample from the pumpkin dataset

Table 4.1: Different scenarios that affect the appearance of apples in a training dataset

Case	Image	Description
1		Apple is partially occluded by a leaf; we include the leaf in the image so the model can learn apples with some leaf features
2		Half of the apple is occluded by a leaf; we include half the apple so the model can recognize what half an apple looks like and avoid over-training the model to start mistaking leaves for apples
3		Apple is heavily occluded by leaves, we only include the visible part of the apple, improving the models ability to recognize partial apple features
4		Apple is overlapping with another apple; though this problem is normally mended as the video progresses and the angle changes, we still want to ensure that the model is able to identify different apples detections that intersect with each other
5		Apples under different lighting: we address the variance in natural lighting where some spots are bright, and others are very dim by including apples under different lighting conditions

and as such, the model learned not to detect them. This was also observed in our initial experiments we've explained previously. Table 4.4 shows the results of running our pipeline on two other different fruits: oranges and pumpkins. This experiment is intended to prove the versatility of our pipeline and the functionality of our modification to DeepSORT. For oranges, we use the pretrained YOLO weights. This was not possible for the pumpkins as well as the pumpkin class does not exist in the COCO dataset. As such, fine tuned weights on pumpkins are used. The L1 Loss between the predicted and ground truth represents the accuracy criteria for model evaluation. The following equation computes the accuracy, where GT is the ground truth and L is the computed loss.

$$Accuracy = \frac{GT - L}{GT} * 100$$

4.3.1 Apples

Initial Experiments on Apple Trees

As seen in Table 4.2, using just YOLO's pretrained weights lead to a significant over count of 181 apples. This is due to how YOLO was trained on a massive dataset (MSCOCO) that features apples of various types and sizes. In this experiment, we observed that distant apple trees have a few apples detected and tracked. At first, we aimed to solve the problem by increasing the minimum confidence needed for the detection to be tracked. This is because the distant detections had very blurry quality and all had low confidence levels. However, that didn't improve the result as there are other cases where the apple detection had lower confidence due to being occluded by other objects or, more commonly, blurred due to camera motion. Thus, the correction mechanism needs to specifically target distant apples to avoid miscounting true apples in the current row. To find a balanced and efficient solution, we ran the detection model on the same video and saved

all detections into a file to determine the average size of a distant apple so we can filter them out. This experiment suggests that a size of 30 pixels width and height for the apple is an appropriate size threshold to filter out distant apples.

Using YOLO's pretrained weights with the correction mechanism yielded a substantially better result, where the model under-counted by 43 apples, giving an accuracy of 87.43%. We analyzed how the detection is behaving and observed that while distant apples are no longer detected, the model sometimes struggles with occluded apples. The detector itself also yielded fair average confidence levels due to using a low confidence. However, the model would perform worse with higher minimum confidence levels. To address these issues, we use transfer learning and fine tune YOLO's weights using the dataset discussed previously, in which we included occluded apples and varying shading. We also changed the configuration of YOLO to only include the apple class instead of MSCOCO's 80 classes. Our fine tuned weights showed an improvement, with the model under-counting by only 29 apples and the accuracy increased to 91.5%. Upon closer investigation of this experiment, we found more occluded apples being detected appropriately and the tracker was able to successfully identify and keep track of such detections. Figures 4.3, 4.4 and 4.5 illustrate the detection and tracking throughout three frames. More importantly, we learned that our correction mechanism is no longer needed as we completely avoided annotating distant apples, which allowed us to remove the extra computation. However, there are still some largely occluded apples which could not be detected throughout all the video frames. Thus, apples that are almost completely hidden, despite any change in angle or lighting as the camera moves, pose a challenge to our pipeline, and in fact to any pipeline.

Experiments on Full Rows of Apple Trees

In our initial experiments on apple trees, we focus on small segments of the trees (with roughly 300 apples) to try and catch as many different cases of apple visibility as possible.

Table 4.2: Results of the proposed pipeline running on a video clip of apples. We show the predicted count versus the actual count and compute the L1 Loss and accuracy. The accuracy is low with the pretrained weights due to a significant overcount. Our correction mechanism substantially improve the accuracy, however fine tuning the weights led to the best performance.

Metrics	Pretrained Weights	Pretrained & Corrected	Finetuned Weights
Predicted/Ground Truth	523/342	299/342	313/342
L1 Loss	181	43	29
Accuracy	47.06%	87.43%	91.5%

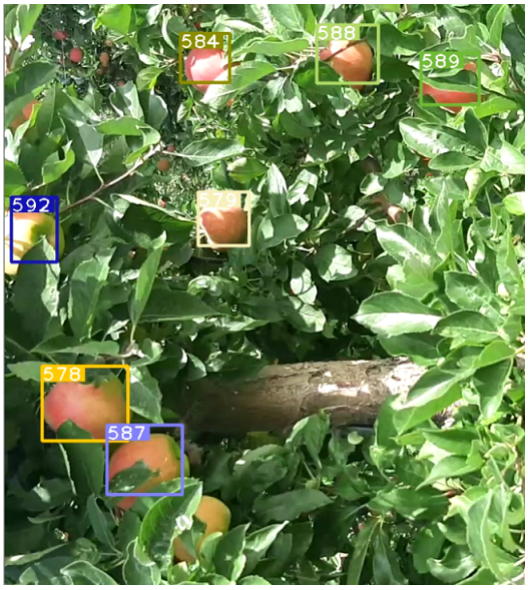


Figure 4.3: Apples with low visibility or occlusion are successfully detected.

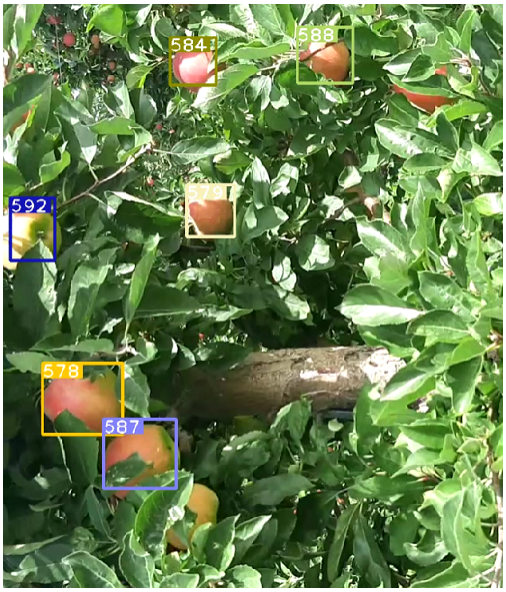


Figure 4.4: The detected apples maintain their track IDs so long as they're detected. Apple ID 589 is not detected in this frame, thus is not shown

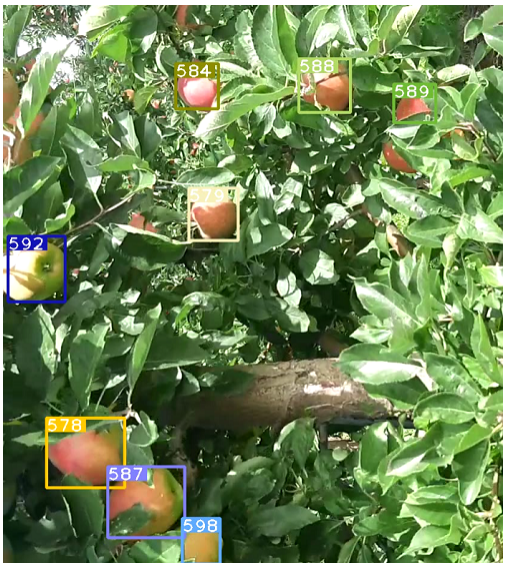


Figure 4.5: The change in angle allows the detector to detect a previously missed apple (ID 598). In addition, the tracker is able to re-identify apple ID 589 when it is detected again

We then expand the use of the framework with much larger and more realistic segments of apple trees (with roughly 3-4k apples) to test the scalability and practicality of the framework. In this experiment, we perform counting on 3 separate full-length rows of apples, catching the whole trees. As our primary goal is to test for scalability, we train our model first with imagery of full apple trees and run our experiments using exclusively the fine tuned weights. Our findings are presented in Table 4.3. Despite each row has different lightnings due to the direction of sunlight and the fact that trees themselves have subtle differences in their shape and density, the result has a little variance ranging between 90.6% to 95.8% and the accuracy closely resembles our initial experimentation that yielded a 91.5% accuracy with fine tuned weights. We inspected each video analysis to understand the reason behind the difference in accuracies. We found that in apple row 1, there are a few trees that are extremely dense on leaves and their apples can barely be noticed, thus a large number of apples are completely undetected in comparison to other rows. For apple row 2, the sunlight was striking the camera directly, which makes all the apples appear darker than they are which presents a visual problem that affects the detector's performance. For this row, we observed that our annotating approach and diversifying the dataset with different conditions as discussed in the previous section have improved the accuracy to 93%, otherwise it could have been much lower. Finally, apple row 3 had good lighting and most of the apples are clearly viewed within the trees, thus the framework achieved an accuracy of 95.8%. This shows that, despite the significantly larger amount of apples per frame, such as in Fig. 4.6, the framework remains robust and efficient. Similarly, visual challenges, such as apples being occluded by leaves, can be observed in Fig. 4.7. As the camera moves and more of the apple pixels get revealed, the apple is detected and appropriately tracked as seen in Fig. 4.8. However, like our previous experiments, largely obscured apples pose a challenge to our framework due to our reliance on detectors to accurately pick up all visible apples. Depending on the density of the tree, as well as farming practice (i.e., whether the tree is pruned or not), this

challenge can either lead to insignificant accuracy loss, or more noticeable under-counts. From this point on, more training and a larger dataset would lead to small increases in accuracy. However, we need to avoid overfitting and ensure that detectors can still pick up subtle differences between each different row in the orchard. Further, our modified DeepSORT algorithm enabled us to only retrain the selected object tracker (in our case, YOLOv3) and avoid retraining or modifying the tracker.

Table 4.3: Results of the proposed pipeline running on a video clip of 3 neighboring rows of apples. We observed consistent performance across the three rows, with accuracy varying between 90-95%. This is consistent with the performance shown on the smaller scale apple detection in the earlier experiment.

Metrics	Apple Row 1	Apple Row 2	Apple Row 3
Predicted/Ground Truth	4375/4827	3647/3921	3530/3683
L1 Loss	452	276	153
Accuracy	90.6%	93.0%	95.8%

4.3.2 Other Fruit Counting: Oranges and Pumpkins

This section reports the experimental results and performance analysis of the proposed framework applied to other fruits (orange and pumpkin specifically) to prove its generalization and feasibility in precision agriculture. Table 4.4 presents the results of our experiments on videos of oranges and pumpkins. Pumpkin footage is captured by a drone, which provides another interesting perspective of applying the framework on far aerial views. Fig. 4.9 shows a frame from the footage that displays the aerial view and current detections and tracks. It can be noticed that very few pumpkins are not yet tagged. There are two common scenarios that might occur in this use case: (1) pumpkins that are not visible in one frame will eventually become more visible in future frames and are appropriately detected and tagged. This is a very common scenario and is shown in Fig. 4.11 and Fig. 4.12. (2) There are few pumpkins that can hardly be seen and are never

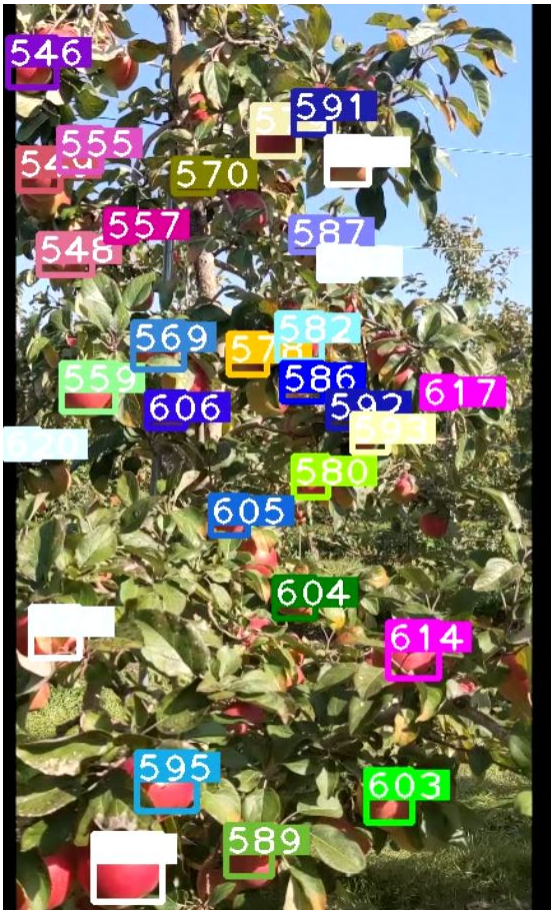


Figure 4.6: A frame taken from the video of the apple tree during runtime, just in this frame there are approximately 30 apples being tracked, in addition to the saved tracks that are not currently detected. There are several apples that are largely occluded for which one of the following scenarios could be true: (1) previously detected and counted before becoming obscured; (2) will be detected next with the camera motion or with a clearer angle; (3) will fail to be detected leading to a loss in counting accuracy.

Figure 4.7: The leaf covers the apple and is predominantly visible. We avoid annotating such apples to avoid mistakenly detecting leaves as apples and will instead rely on the angle eventually making the apple clearer.

Figure 4.8: The apple does indeed become clearer in the following frame, allowing for detection to occur and the tracker to save and count the apple.

detected, thus are never tagged by the tracker, such as in Fig. 4.10. This is similar to the cases in apple counting. During our pumpkin experiments we faced major issues in finding video data with pumpkins and used a relatively limited number of short videos and images to train our model. Despite this limitation, our modified DeepSORT tracker function near-perfectly with the detections provided and achieves a high accuracy of 94.9%. Due to the formation of the pumpkin patch, it's a straightforward task to detect each individual pumpkin from the top-down view that the drone provides. Additionally, pumpkins have a distinct and relatively large shape and its bright orange color stands out from the surrounding background (grass and leaves). On the other hand, the base YOLO model already has an oranges class and the default pretrained weights are trained to recognize oranges. As such, we use the pretrained weights for the detector to show that the modified DeepSORT tracker can efficiently run on any object without further training or modification. However, while the tracker works with oranges and produces a count, as seen in Fig. 4.13, the accuracy is lagging a little bit, showing an accuracy of 79.3%. An analysis of the footage during runtime quickly shows oranges that are heavily occluded by other oranges or branches, as seen in Fig. 4.14, failed to be detected even throughout the movement of the camera. This problem existed in our earlier analysis on apples using pretrained weights, however, our data augmentation and annotation approach explained in Section 4.2 tackle this challenge very well and lead to higher accuracy. Unfortunately, collection of real orange footage is challenging due to the limitations of orange orchards in our area and lack of such footage or available data online.

Table 4.4: Results of the proposed pipeline running on a video clip of pumpkins and oranges. The oranges are counted using pretrained YOLO weights and thus produce a lower accuracy of 79.3%. Since pumpkins are trained specifically on aerial views of pumpkins, including a sample from the experiment video, the accuracy was quite high.

Metrics	Pumpkin Counting	Orange Counting
Predicted/Ground Truth	219/233	96/121
L1 Loss 1	14	25
Accuracy	93.9%	79.3%

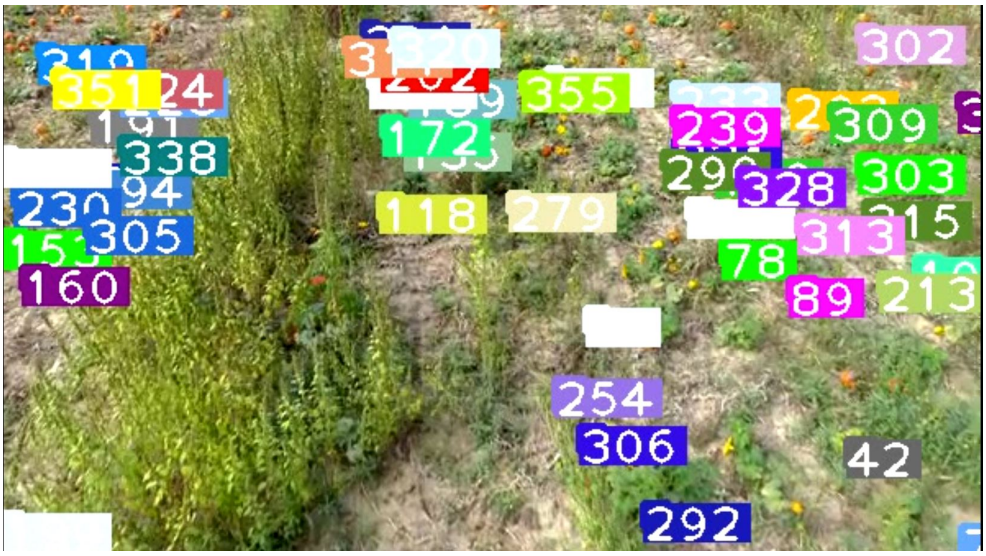


Figure 4.9: A frame taken from the video showing the view of the pumpkins and all of the current detections, the numbers denote the track ID.

4.4 Container Placement Results

We apply our container placement algorithm on apple row 3 from our second experiment. While the feed of apple row 3 undergoes counting, the GPS coordinates are synchronized and annotated with the counts as discussed in Section 3.4. After the yield estimation process is complete, we process the geospatial and count information to begin proposing the number of required containers and their optimal placements. We apply different sets of constraints when it comes to the maximum weight of the container, and the maximum distance between each placement. We included containers that can occupy up to 300

Figure 4.10: Pumpkin is mostly hidden and is hard to be seen due to little to no lighting, in further frames the pumpkin only becomes more hidden and is never detected.

Figure 4.11: Another pumpkin that's hidden and is not currently detected nor counted.

and 1000 apples and set our maximum distance to 40 feet and 100 feet . In addition to checking the location and number of containers, we also check the utilization of a container. For example, if 1000 apples are assigned to a container of size 1000, this means the container is 100% utilized and we achieve the minimum number of containers by having as many fully utilized containers as possible. We show the container placement locations and container utilization in Tables 4.5, 4.6 and 4.7. Note that the pair (300 apples and 100 feet) yielded the same result as when the distance was set to 40.

We can observe that in the case where the maximum distance was set to 100 feet, all of the containers were fully utilized relative to the apples being assigned regardless of the maximum capacity. However, there are some concerns that with larger sized containers (possibly even greater than 1000), there will be too much distance between the harvester and the container in which they collect the apples. When we set the maximum distance

Figure 4.12: The change in view as the drone flies forward allows more of the pumpkin to be seen, thus is successfully detected and given a track ID.

to 40 feet instead, we note a more uniform distribution of the apples across the containers as seen in Table 4.6. This also reflects on the mapping of the containers seen in Fig. 4.15, where containers have more even spacing when the distance constraint goes into effect as opposed to Fig. 4.16. We conducted additional testing with larger size containers of 2000 apple capacity, we found that the container assignment is the exact same as when the container size is 1000 when the maximum distance is 40 feet. Whereas only two containers are placed when the distance is set at 100 feet. We conclude that it's up to the farmers discretion to make the final tradeoff between far spaced and lesser number of containers, or tightly spaced containers that are easy to reach for the harvester based on our proposed assignments and visualization.

4.5 Summary

In this chapter, we demonstrate the effectiveness of the proposed framework by performing experiments on different types of fruits. We define a novel strategy for collecting and annotating fruit datasets to specifically address visual challenges that are common within fruit trees. The framework has been tested on apples, pumpkins and oranges. Performance evaluation shows that the framework achieves up 95% accuracy on apples and

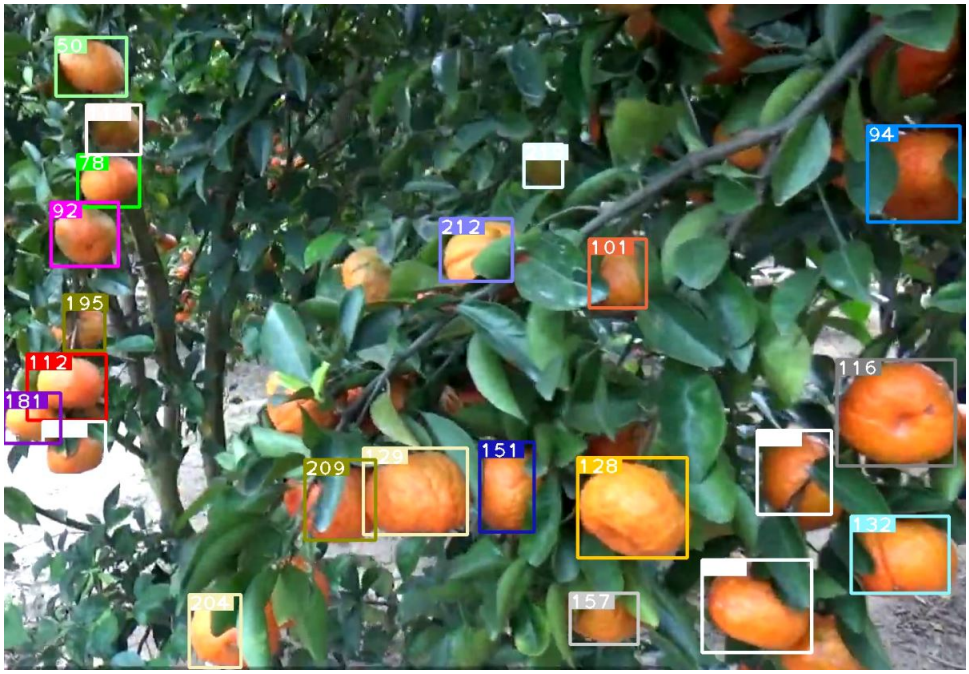


Figure 4.13: A view of detected oranges in the tree, numbers denote track ID.

pumpkins, and 79+% on oranges. Lastly, we present an evaluation of our container placement solution with different container sizes, different distances and thresholds between containers.

Figure 4.14: The majority of uncounted oranges are heavily obscured behind other oranges and leaves, the YOLO pretrained weights don't fully accommodate brightness and occlusion challenges.

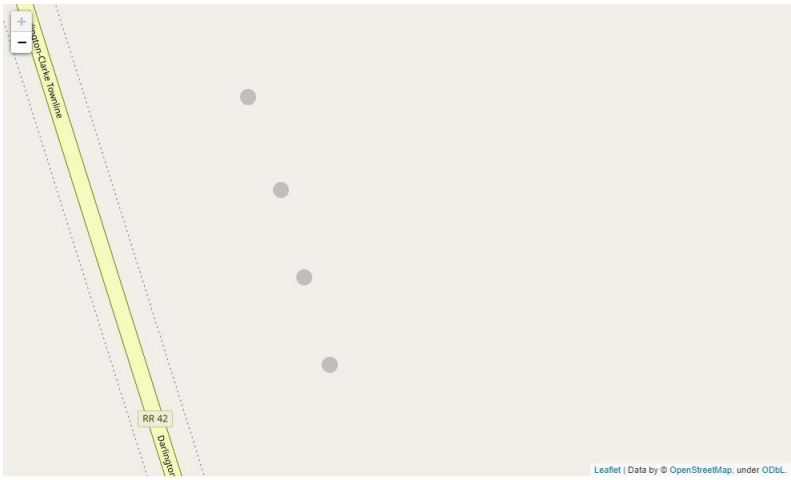


Figure 4.15: The container placements visualized using Folium and OpenStreetsMap template. The distance between the containers is evenly spaced across the row, ensuring harvesters will have a container near them.

Figure 4.16: The distance between the containers is uneven, with the last two containers being close to one another. This means that harvesters between the 2nd and 3rd boxes will walk longer distances. There might also be a crowd around the 3rd and 4th boxes as they are fairly close to one another.

Table 4.5: All of the assigned containers are fully utilized. Note that while the last container has 77% utilization, this is because the remaining number of apples was 230 at that point, not 300, so 77% is the maximum utilization the container can reach.

Latitude	Longitude	Utilization of Container
43.91716	-78.62771	100%
43.91732	-78.62776	100%
43.9174	-78.62782	100%
43.91757	-78.62788	100%
43.91774	-78.62796	100%
43.91786	-78.628	100%
43.91805	-78.6281	100%
43.91818	-78.62814	100%
43.91833	-78.62821	100%
43.91853	-78.62829	100%
43.91865	-78.62834	100%
43.91872	-78.62838	77%

Table 4.6: None of the containers have 100% utilization due to the maximum distance restriction, however they're still fairly highly utilized, thus no containers are wasted and the farmer may find this to be a favorable balance between even spacing of containers and properly utilizing the container capacities.

Latitude	Longitude	Utilization of Container
43.91743	-78.62783	97.4%
43.91785	-78.628	80%
43.91827	-78.62816	76.7%
43.91872	-78.62838	98.9%

Table 4.7: The containers are fully utilized, however the last container is only half full and is placed too close to the 3rd container, and the other three containers have high spacing between them. This is a less favorable option for the farmer as it adds extra time and effort for the harvesters.

Latitude	Longitude	Utilization of Container
43.91745	-78.62783	100%
43.91798	-78.62807	100%
43.91853	-78.62829	100%
43.91872	-78.62838	53%

Chapter 5

Conclusion

We present our framework for yield estimation and visualization using deep learning for use in precision agriculture. The framework essentially follows through three stages after data collection. First, the video is inserted into an object detector, frame-by-frame, where the detector draws bounding boxes around every fruit in the scene and classifies it. Afterwards, the detections are passed on to a tracking algorithm that we modified based on the widely used DeepSort algorithm. The tracker is capable of drawing associations between detected fruits in each frame, meaning that if fruit x appears in frames 1 and 2, the tracker is able to uniquely match the detections of fruit x and track its movement through future frames. In the last stage of our framework while tracking the fruit, a count is deduced and geospatial data is incorporated with the count in order to map the yield on a visual map. Mapping the yield aids in making decisions in regards to harvest planning. In our work, we also present a container placement algorithm which leverages the count and geospatial data to suggest an efficient strategy for container positions.

5.1 Discussion

We have demonstrated the effectiveness of our framework for fruit counting in three different cases: apples, oranges and pumpkins, which can be seen in the results provided in Section 4.1. The tested datasets are widely diverse, in the case of the apples we tested both simple views of apple trees, and full-scale footage of entire rows of trees to show the scalability of the proposed framework. We also included other fruits, with video footage of an orange tree, and an aerial view of a pumpkin patch to test our framework's versatility. Because we implemented the ResNet18 model instead of the original proposed CNN model, we turned the tracker into a more effective generic solution for tracking different classes. To expand, the original proposed model is only trained on people tracking and as such we need to retrain it from scratch for every specific use case, which is a very complex and time consuming process. By using ResNet18, we not only use a model capable of

identifying 1000 classes, but we are also guaranteeing a well performing model as ResNet is one of the best established CNN models in research. As a product of our modification, whether the fruit is apple, pumpkin, orange or anything else, we do not need to change or train the tracker, instead we just need to train the object detector.

A secondary point we find important is the proper annotation of our fruit datasets when training the object detector. In computer vision, a vital aspect to the success of deep learning models is accurate annotation of training datasets. Specifically, when annotating for object detection datasets, a bounding box must be drawn around every object of interest. The boxes must be tight to cover the entire view of the object, but capture as little noise (from the background or other occluding objects) as possible. Though this process is time consuming, it's the primary reason why deep learning-based object detectors are highly functional. In essence, the model is trained to recognize any patterns and associate them with the annotated objects. Following this philosophy, we propose specific annotation strategies when it comes to annotating fruit data, and specifically, our apple data. We found that by including views of apples in different scenarios, such as when obscured by a tree, when covered by another apple, or when in varying brightness, the accuracy of our detector improved significantly.

In our work, we integrate geospatial information, specifically GPS coordinates, with the produced yield estimation. We found that many farmers have keen interest in seeing more about their field and how their yield is distributed and where to optimally place containers. We proposed an approach to matching GPS coordinates to frames so that the count within the frames is tagged with a specific GPS coordinate. This allowed us to visualize the GPS coordinates annotated by the respective count in that specific location. By extension, we then leverage the mapping to present a smart harvesting solution, where we developed a container placement algorithm to recommend positions to place containers, according to the fruit count in each specific area and user-defined constraints such as minimum distance between containers and capacity. We approach

container placement as an optimization problem, where we aim to minimize the number of containers needed while ensuring no fruit is assigned to more than one container, and that distance between containers has a threshold to avoid massive distance in patches where the yield is low. We found that the integration of GPS coordinates with yield estimation enables smart applications for better logistics management and efficient harvesting, such as our successful container placement.

5.2 Future Work

The possibilities to expand on the research and applications of deep learning and computer vision in precision agriculture are endless. For our work specifically, a logical next step is to incorporate newer object detection models that theoretically should outperform YOLOv3 in both accuracy and performance speed. This is because the primary limitation of our framework is detection. If an object is not accurately detected, it won't be tracked properly and thus not counted. Expanding the sizes of our datasets would also improve the overall performance of our framework. Limitations when it comes to denser trees, where an abundance of apples are completely hidden within a tree, can potentially be addressed using machine learning (e.g., linear regression) in combination with computer vision to give precisely accurate yield estimates.

Another point that could be of value to farmers is to include more crop analysis into the framework. For example, research for identifying fruit diseases that affect fruit ripeness seem promising, with some solutions recommending training a CNN classifier to identify various diseases that could potentially affect a plant, since many symptoms can have identifiable visual effects. There is some research potential in developing techniques that would not require training a model from scratch on all kinds of diseases. The classification of diseases, in combination with the geospatial technique we used in our work, can help the farmer identify exactly where diseases might be spreading, and act

accordingly. The same paradigm also extends to weeds, which is a very common problem across all kinds of agricultural fields.

Geospatial information can also be used for path optimization of agricultural robots. Robots tend to operate on limited energy, and by forming an optimal path, we can increase efficiency to either aerial or ground robots that are used for irrigation, treatment dispensing, and other field tasks.

5.3 Conclusion

This work on yield estimation and visualization using deep learning develops an end-to-end framework that receives video footage of fruit trees and produces accurate yield estimates combined with GPS coordinates that are visually mapped. Using video footage, due to their efficiency, demands the addition of a tracking functionality in addition to object detection which is used in past works for yield estimation. In this work, we use YOLOv3 for object detection and the DeepSORT tracking algorithm for tracking. We modified DeepSORT to work more robustly on various fruits by implementing ResNet18 in place of the original proposed CNN. Our modification was successful and the framework performs well on various fruits from different views, producing high accuracy. While collecting apple data from the orchard, we recorded the GPS coordinates. We leverage this data to associate a set of frames with a respective coordinate, allowing us to visualize the yield information on a map. Using geospatial information, we were able to implement an efficient container placement algorithm that suggests optimal locations for containers in preparation for harvesting. Through the inclusion of both visualized yield estimates and optimal container placements, we are able to present farmers with a smart harvesting solution that aids them in understanding the states of their fields, and efficiently plan their logistics before harvest. We found that our work can be limited by the performance of our detector and some visual challenges such as object occlusion. We developed a

strategy for fruit data annotation to tackle these challenges and diversify the detector training dataset for better performance. Although this strategy was successful and has significantly improved the detector performance, the detector remains the bottleneck in the proposed framework. Overall, the proposed framework is successful in producing accurate yield estimates and generating a mapping solution to improve the usability of fruit analysis to farmers, supporting decision making for better and efficient logistics management.

Bibliography

[1] Suchet Bargoti and James P Underwood. Image segmentation for fruit detection and yield estimation in apple orchards. *Journal of Field Robotics*, 34(6):1039–1060, 2017.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[3] Kushtrim Bresilla, Giulio Demetrio Perulli, Alexandra Boini, Brunella Morandi, Luca Corelli Grappadelli, and Luigi Manfrini. Single-shot convolution neural networks for real-time fruit detection within the tree. *Frontiers in plant science*, 10:611, 2019.

[4] YT Chan, AGC Hu, and JB Plant. A kalman filter based tracking scheme with input estimation. *IEEE transactions on Aerospace and Electronic Systems*, (2):237–244, 1979.

[5] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. Hardnet: A low memory traffic network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3552–3561, 2019.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2016.

[7] Steven W Chen, Shreyas S Shivakumar, Sandeep Dcunha, Jnaneshwar Das, Edidiong Okon, Chao Qu, Camillo J Taylor, and Vijay Kumar. Counting apples and oranges with deep learning: A data-driven approach. *IEEE Robotics and Automation Letters*, 2(2):781–788, 2017.

[8] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 2, pages 142–149. IEEE, 2000.

[9] DL Corwin and SM Lesch. Application of soil electrical conductivity to precision agriculture: theory, principles, and guidelines. *Agronomy journal*, 95(3):455–471, 2003.

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

[11] Simon Denman, Vinod Chandran, and Sridha Sridharan. An adaptive optical flow technique for person tracking systems. *Pattern recognition letters*, 28(10):1232–1239, 2007.

[12] Xianzhi Du, Tsung-Yi Lin, Pengchong Jin, Golnaz Ghiasi, Mingxing Tan, Yin Cui, Quoc V. Le, and Xiaodan Song. Spinenet: Learning scale-permuted backbone for recognition and localization, 2019.

[13] Ross Girshick. Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 Inter:1440–1448, 2015.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation (R-CNN). *Proceed-*

ings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 580–587, 2014.

[15] Elisabetta Giusti and Stefano Marsili-Libelli. A fuzzy decision support system for irrigation and water conservation in agriculture. *Environmental Modelling & Software*, 63:73–86, 2015.

[16] A Gongal, Suraj Amatya, Manoj Karkee, Q Zhang, and Karen Lewis. Sensors and systems for fruit detection and localization: A review. *Computers and Electronics in Agriculture*, 116:8–19, 2015.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[18] Xiaobing Han, Yanfei Zhong, Liqin Cao, and Liangpei Zhang. Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification. *Remote Sensing*, 9(8):848, 2017.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397, 2017.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8691 LNCS(PART 3):346–361, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Lecture Notes in Computer Science*, page 346–361, 2014.

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

[23] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In *European conference on computer vision*, pages 749–765. Springer, 2016.

[24] Mahbub Hussain, Jordan J Bird, and Diego R Faria. A study on cnn transfer learning for image classification. In *UK Workshop on computational Intelligence*, pages 191–202. Springer, 2018.

[25] Glenn Jocher, Yonghye Kwon, guigarfr, perry0418, Josh Veitch-Michaelis, Ttayu, Daniel Suess, Fatih Baltacı, Gabriel Bianconi, and IlyaOvodov. Ultralytics yolov5 release compatibility update for yolov3. 2021.

[26] A Koirala, KB Walsh, Z Wang, and C McCarthy. Deep learning for real-time fruit detection and orchard fruit load estimation: Benchmarking of 'mangoyolo'. *Precision Agriculture*, 20(6):1107–1135, 2019.

[27] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178, 2006.

[28] Bastian Leibe, Konrad Schindler, and Luc Van Gool. Coupled detection and trajectory estimation for multi-object tracking. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[29] Xin Li, Kejun Wang, Wei Wang, and Yang Li. A multiple object tracking method using kalman filter. In *The 2010 IEEE international conference on information and automation*, pages 1862–1866. IEEE, 2010.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[31] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection (FPNs). *Cvpr*, 3(3):137–147, 2017.

[32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[33] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[34] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path Aggregation Network for Instance Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.

[35] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 385–400, 2018.

[36] Songtao Liu, Di Huang, and Yunhong Wang. Learning spatial fusion for single-shot object detection. *arXiv preprint arXiv:1911.09516*, 2019.

[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS:21–37, 2016.

[38] Xu Liu, Steven W Chen, Shreyas Aditya, Nivedha Sivakumar, Sandeep Dcunha, Chao Qu, Camillo J Taylor, Jnaneshwar Das, and Vijay Kumar. Robust fruit counting: Combining deep learning, tracking, and structure from motion. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1045–1052. IEEE, 2018.

[39] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[40] Zeeshan Malik, Sheikh Ziauddin, Ahmad R Shahid, and Asad Safi. Detection and counting of on-tree citrus fruit for crop yield estimation. *Int. J. Adv. Comput. Sci. Appl*, 7(7), 2016.

[41] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.

[42] Kemal Oksuz, Baris Can Cam, Sinan Kalkan, and Emre Akbas. Imbalance problems in object detection: A review. *arXiv*, 2019.

[43] Cle Pohl and John L Van Genderen. Review article multisensor image fusion in remote sensing: concepts, methods and applications. *International journal of remote sensing*, 19(5):823–854, 1998.

[44] Zania S Pothen and Stephen Nuske. Texture-based fruit detection via images using the smooth patterns on the fruit. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5171–5176. IEEE, 2016.

[45] Maryam Rahnemoonfar and Clay Sheppard. Deep count: fruit counting based on deep simulated learning. *Sensors*, 17(4):905, 2017.

[46] M Narayana Reddy and NH Rao. Gis based decision support systems in agriculture. *National Academy of Agricultural Research Management Rajendranagar*, pages 1–11, 1995.

[47] Joseph Redmon. Darknet: Open source neural networks in c, 2013.

[48] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

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

[50] Pravakar Roy and Volkan Isler. Surveying apple orchards with a monocular vision system. In *2016 IEEE international conference on automation science and engineering (CASE)*, pages 916–921. IEEE, 2016.

[51] Inkyu Sa, Zongyuan Ge, Feras Dayoub, Ben Upcroft, Tristan Perez, and Chris McCool. Deepfruits: A fruit detection system using deep neural networks. *Sensors*, 16(8):1222, 2016.

[52] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks, 2014.

[53] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation (FCNs). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition (VGG Net). *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–14, 2015.

[55] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[56] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[57] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[58] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[59] Lei Tian, John F Reid, and John W Hummel. Development of a precision sprayer for site-specific weed management. *Transactions of the ASAE*, 42(4):893, 1999.

[60] Shaohua Wan and Sotirios Goudos. Faster r-cnn for multi-class fruit detection using a robotic vision system. *Computer Networks*, 168:107036, 2020.

[61] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.

[62] Qi Wang, Stephen Nuske, Marcel Bergerman, and Sanjiv Singh. Automated crop yield estimation for apple orchards. In *Experimental robotics*, pages 745–758. Springer, 2013.

[63] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[64] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[65] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

[66] Yang Yu, Kailiang Zhang, Li Yang, and Dongxing Zhang. Fruit detection for strawberry harvesting robot in non-structural environment based on mask-rcnn. *Computers and Electronics in Agriculture*, 163:104846, 2019.

[67] Sungmin Yun and Sungho Kim. Recurrent yolo and lstm-based ir single pedestrian tracking. In *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, pages 94–96. IEEE, 2019.

[68] Hua Zhang, Shanzhen Yi, and Yonggang Wu. Decision support system and monitoring of eco-agriculture based on webgis in shule basin. *Energy Procedia*, 14:382–386, 2012.

[69] Kun Zhao and Wei Qi Yan. Fruit detection from digital images using centernet. *Geometry and Vision*, 1386:313, 2021.

[70] Liang Zheng, Zhi Bie, Yifan Sun, Jingdong Wang, Chi Su, Shengjin Wang, and Qi Tian. Mars: A video benchmark for large-scale person re-identification. In *European Conference on Computer Vision*, pages 868–884. Springer, 2016.

[71] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.