AUTOMATIC DETECTION AND SEGMENTATION OF GREENHOUSE FRUIT USING DEEP NEURAL NETWORKS

by

Yan Du

(Under the Direction of Yi Hong)

ABSTRACT

Accurate detection and segmentation techniques are essential to monitor fruit crop growth and perform yield estimation automatically. Recent research on deep neural networks such as Faster R-CNN and Mask R-CNN has shown promising results on automatic fruit detection and segmentation. However, these networks suffer problems of false positives and missing fruits, while these hard examples often exist in fruit images collected in greenhouses due to leaf occlusion and poor lighting conditions. This thesis proposes to augment Faster R-CNN and Mask R-CNN with an online hard example mining (OHEM) algorithm, resulting in considerably improved detection and segmentation results on our dataset. The Mask R-CNN with OHEM jointly detects and segments tomatoes and demonstrates the best performance with an F1-score of 0.957 for detection and a dice sore of 0.816 for segmentation. Applications of the proposed networks include tomato counting and growth monitoring, suggesting the promise of their future deployment in greenhouses.

INDEX WORDS: Object Detection, Image Segmentation, Faster R-CNN, Online Hard Example Mining, Mask R-CNN

AUTOMATIC DETECTION AND SEGMENTATION OF GREENHOUSE FRUIT USING DEEP NEURAL NETWORKS

by

Yan Du

BS, Shandong University, China, 2011

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2018

AUTOMATIC DETECTION AND SEGMENTATION OF GREENHOUSE FRUIT USING DEEP NEURAL NETWORKS

by

Yan Du

Major Professor:	Yi Hong
Committee:	Khaled Rasheed
	Frederick Maier

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
August 2018

DEDICATION

Firstly I would like to thank Dr. Hong as a great mentor during my research time. Her enthusiasm on research did encourage me a lot. I have learned a lot in writing from her as well. Secondly, I want to thank my parents, my husband, and my parents in law for their unconditional support, especially during the past three years. Thirdly, I would like to thank my lab mates, Parya, Raunak, Shuhbi, Sharmin, Dhara, etc. Thank you guys for your help. Sometimes I feel more clear in my problem after doing the brain-storming with you. Also I want to thank Bahaa for your patient guidance on the installation of some deep learning frameworks.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my major professor, Yi Hong, for her support and assistance throughout the course of my studies. I would also like to extend my thanks to the members of my thesis committee, Dr. Rasheed and Dr. Maier, for their time and service.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 **Background**

Fruit harvesting has been playing an increasingly important role and becoming a very critical process in the overall fruit economics. For the fruit to bear quality fruit for a longer life-cycle, the fruit need to be harvested at the right time, i.e. right maturity. By not harvesting at the right stage, the total yield per plant or farm can be greatly reduced, affecting both economy and daily supply negatively. Successful fruit harvesting requires accurate monitor of the growth of the fruit. Being able to precisely monitor fruit growth is highly desired in the fruit industry because it is of tremendous importance in identifying abnormal growth, estimating the overall yield and arranging the final fruit harvesting. However, conventional approaches by sourcing skilled farm labors in the agriculture industry is one of the most cost-demanding factors due to the rising values of supplies [1]. Therefore, to meet the growing demands of an ever-growing world population and cater to fresh markets, intelligent approaches that enables accurate localization of the fruit and automatic surveillance of fruit growth have been a long-standing request in the fruit industry.

Toward this goal, one promising alternative is automatic fruit harvesting, central to which is a fruit detection system that allows to perform yield estimation and mapping by providing an accurate localization of individual fruit in the field. Therefore, the development of an accurate fruit detection system is key to successful automatic harvesting. In general, the main objectives of fruit detection include two components, including 1) to distinguish the fruits from the

background; 2) to localize the fruits. Upon accurate fruit detection results, segmentation can then be carried out to classify each pixel into different classes.

Previous works took advantages of hand-engineered features to represent the visual cues for discriminating the fruit from the background [2]. Despite that these approaches are well suited for the dataset they are designed for, feature encoding is usually unique to a specific fruit and the conditions under which the data were captured [3]. Therefore, changing the fruit for detection from one to another usually requires repeated hand-engineered encoding of the features of the new fruit, representing a major limitation for these approaches.

Deep Neural Network (DNNs) offers a perfect solution to the above mentioned problem by providing automatic feature extraction for object detection and image semantic segmentation which avoids the need of hand engineered features. One of the state-of-the-art deep learning detectors, Faster R-CNN [1], has been recently employed by many fruit detection systems to improve the detection performance. For example, Sa et al. [4] proposed a fruit detection system, which applied Faster R-CNN framework. Their method has achieved detection of six different fruits, and improved accuracy compared to prior work, which in principle infers accurate detection of any fruit species.

Being able to monitor the growth of fruit using DNN based methods will eventually help to achieve two major goals both of which are very difficult and inefficient to accomplish using conventional labor-based approaches. The first goal lies in achieving maximum growth rate while lowering necessary energy consumption, an outstanding challenge in agriculture industry. Given that greenhouse offers a well-controlled environment for fruits, performing fruit detection in a greenhouse provides us unique opportunity to monitor the growth rate as a function of environment conditions, such as lighting intensity and temperature. This can be studied by

adjusting the lighting conditions and controlling the temperature while monitoring the growth rate of the fruit. The other goal is associated with quality control and fruit production. By monitoring the growth of every individual fruit, one can easily check fruit health and find out abnormal ones, which leads to better estimation of fruit quality and the final production.

The reason for choosing tomato as the detection fruit in my thesis is the following: Tomato is one of the most consuming fruits in the market and is easily available and cheap for analysis purpose. Tomato is also a major horticulture accommodated in greenhouse, offering a suitable environment for fruit detection task [5]. It has been shown that its demand sometimes increases at very high level in market due to low production or damage due to disease attacks [6]. Therefore, there is an essential need of an advance system for tomato detection.

1.2 Motivation

Accurate counting of tomatoes and monitoring of their growth rate allow farmers to track the tomatoes over time, plan maintenance and harvesting activities, and further predict the yield, a crucial step of the overall agricultural industry. To realize precise tomato yield estimation, fully automated approaches utilizing object detection techniques have been confirmed suitable and employed in various detection tasks to tackle such problems.

To monitor the growth rate of tomatoes, fruit size can be measured by applying segmentation for each instance. Segmentation approaches provide the number of pixels within the segmented mask for each instance. One is, therefore, able to measure the relative size of the tomatoes in the image, offering tomato size, i.e., maturity, for better arrangement of following activities such as harvesting.

Beyond the demanding need for automatic detection techniques for agricultural activities, the object detection community has been driven by the search for more accurate detection

method. One of the challenges for object detection approaches (e.g., Faster R-CNN) is that the datasets contain a large number of easy examples and a small number of hard examples. This challenge motivates a few learning techniques that deal with hard examples. For instance, Shrivastava et al. [7] presented an effective Online Hard Example Mining (OHEM) algorithm for training region-based CNN detectors. Experimentally, OHEM combing with Fast R-CNN leads to state-of-the-art detection results by eliminating many heuristics and hyper-parameters. Therefore, it is intuitive to propose a framework to augment Faster R-CNN with OHEM for tomato detection.

On the other hand, in order to achieve more accurate detection results, many approaches perform joint object detection and image segmentation. One of these approaches, Mask R-CNN [8], outperforms the base variants of all previous state-of-the-art models on object detection results. Therefore, it is also reasonable to propose a framework to augment the Mask R-CNN with OHEM for tomato detection and segmentation.

1.3 Contributions

In my thesis work, I firstly apply the widely-adopted Faster R-CNN on several fruit datasets including apple, mango, and tomato. Secondly, as a new method developed in this thesis, Faster R-CNN is combined with Online Hard Example Mining (OHEM) algorithm to tackle the hard examples on feature level. Thirdly, I also apply the commonly-used Mask R-CNN only on tomato datasets. At last, I establish a new approach in the present thesis where Mask R-CNN is augmented with OHEM. Moreover, cross validation is performed to evaluate all approaches. A T-test is applied to compare different approaches. The performance of detection and segmentation improves after augmenting Faster R-CNN with OHEM and augmenting Mask R-

CNN with OHEM. However, the T-test results suggest that the improvement is not significant. Mask R-CNN significantly outperforms Faster R-CNN in Recall value and F1-score based on the T-test. The main contributions are:

- Deploying and adapting the state-of-the-art object detection frameworks, Faster R-CNN and Mask R-CNN, to detect tomatoes and measure their growth rates,
- Combing Faster R-CNN with OHEM to tackle with hard examples on the feature level,
- Augmenting Mask R-CNN with OHEM for joint object detection and image segmentation.

The remainder of the thesis is organized as follows. Some related works (e.g., R-CNN, Image Segmentation) are introduced in Chapter 2, followed by a detailed description of the Faster R-CNN with OHEM and Mask R-CNN with OHEM in Chapter 3. In Chapter 4, I present the evaluation metrics along with the experimental results on several fruit datasets. Furthermore, a comparison among different approaches will be given. K-fold cross validation is performed to evaluate the detection and segmentation performance as well. A T-test is performed to calculate the difference of results between Faster R-CNN and Faster R-CNN with OHEM, and the difference of results between Mask R-CNN and Mask R-CNN with OHEM, the difference of results between Faster R-CNN and Mask R-CNN, and the difference of results between Faster R-CNN with OHEM and Mask R-CNN with OHEM, respectively. At last, a summary and future direction are addressed in Chapter 5.

CHAPTER 2

RELATED WORK

To tackle with fruit detection problems, various algorithms that have been developed. This chapter provides a survey of the widely used methods, which are most relevant to fruit detection and segmentation and therefore adopted in my thesis work. This chapter is organized as follows: 1) object detection based on DNNs; 2) image segmentation; 3) fruit detection and segmentation.

2.1 Object Detection Based On DNNs

Object detection determines location and scale of all the instances of objects in the image by outputting a bounding box around the detected object along with the corresponding class label and confidence score. There are several deep learning techniques for object detection task. I will give an overview of the deep learning approaches for object detection in the following sections.

2.1.1 Region-based CNN (R-CNN)

The Region-based Convolutional Neural Network (R-CNN) [9] proposed by Girshick et al. is the first variant using deep Convolutional Neural Networks (CNNs) on bounding box object detection by generating candidate object regions. Uijlings et al. [10] proposed Selective Search for Object Recognition by combining an exhaustive search algorithm and segmentation. Selective Search has been widely used as a proposal method for object detection. The R-CNN uses Selective Search to generate the candidate object regions. Hosang et al. [11] provided a clear analysis of twelve proposal methods. Due to the development in deep CNNs like in [12,

13], the R-CNN utilizes the CNNs on each Region of Interest (RoI). In [14], He et al. extended the R-CNN by introducing a new network architecture called SPP-net. In SPP-net, the features in the feature maps are pooled into regions using Spatial Pyramid Pooling. Moreover, during inference, SPP-net performs much faster than R-CNN, along with higher accuracy on PASCAL VOC 2007.

2.1.2 Fast R-CNN and Faster R-CNN

The other extension of R-CNN is the Fast R-CNN proposed by Girshick [15]. Intuitively, it is called Fast R-CNN because it runs faster speed and yields better accuracy compared to R-CNN. A new pooling strategy called RoIPool is used to pool the RoIs on the feature maps. Faster R-CNN [1] is faster than Fast R-CNN because it involves introducing the RPN to generate the region proposals instead of using Selective Search.

Faster R-CNN contains two modules. The first module Region Proposal Network (RPN), which is a deep fully convolutional network that proposes regions of interest. The second module is the Fast R-CNN detector [15], which is a detection network that utilizes the proposed regions. In a Region Proposal Network, the input is an image, and outputs is a set of object proposals, each with a classification score. This process is modeled with a fully convolutional network, which was proposed by Long et al [16]. Both RPN and Fast R-CNN object detection network share a common set of convolutional layers. As a result, the computation between RPN and Fast R-CNN detection network are sharing.

In the Faster R-CNN implementation [1], the network classifies and regresses bounding boxes with reference to anchor boxes of multiple scales and aspect ratios. Since this multi-scale design is based on anchors, it is easy to use the convolutional features that are computed on a single-scale image, as is done by the Fast R-CNN detector as well.

Lin et al. proposed a method , which is one of the further improvements based on Faster R-CNN. They introduced a new architecture called Feature Pyramid Network (FPN) for building semantic feature maps at different scales. The other improvement was proposed by Huang et al. [17]. They offered a guidance on selecting an object detection system to achieve the best trade-off between speed and accuracy.

2.1.3 You Only Look Once (YOLO)

YOLO [18] is an object detection approach, which models object detection as a regression problem to bounding boxes and corresponding class probabilities. This framework processes the images in real-time at 45 frames per second. Specifically, this approach resizes the input image to 448 × 448, then runs a single convolutional neural network on the images, and finally outputs the resulting detections by providing a threshold. This unified model for object detection is easy to construct and can be directly trained. YOLO achieves higher localization errors but is less likely to predict the false positives on background, compared to other state-of-the-art object detection systems.

2.1.4 Single Shot MultiBox Detector (SSD)

SSD [19] is a fast single-shot object detection approach. It achieves over 74% mAP (mean Average Precision) at 46 frames per second on standard datasets (e.g., PASCAL VOC). Firstly, the single shot means the tasks of object localization and classification are done in a single forward pass of the network. Secondly, the MultiBox is a technique for bounding box regression. Thirdly, the SSD architecture builds on the VGG16 architecture, but without the fully connected layers. SSD produces worse performance on smaller objects, as they may not appear across all feature maps.

Table 1. An overview of several object detection approaches. Note that the results are on PASCAL VOC 2007 test set.

Approach	Region Proposal Generation	Feature Extraction	mAP	FPS	Notation
R-CNN	Selective Search	Deep CNN	66.0%	0.02	Slow at test time
Fast R-CNN	Selective Search	Deep CNN	70.0%	0.5	Need external region proposal approach (e.g., SS)
Faster R-CNN	Region Proposal Network	Deep CNN	**76.4%**	7	Good balance between accuracy and speed
YOLO	Takes the whole image at once instead of region proposals	Deep CNN	63.4%	45	Fast
SSD300	Eliminates proposal generation, with all computation in a single network	Deep CNN	74.3%	**46**	Fastest, but performs worst for small objects

Table 1 shows a comparison between several object detection approaches. More specifically, R-CNN, Fast R-CNN, and Faster R-CNN are region proposal-based approaches. R-CNN and Fast R-CNN use Selective Search to generate the region proposals, while Faster R-CNN uses the Region Proposal Network by sharing the convolutional layers with the detector. Furthermore, Faster R-CNN is the fastest approach at test time without any external region

proposal approaches for region proposal generation. Overall, SSD is the fastest approach, however, Faster R-CNN performs the best trade-off between accuracy and speed.

2.2 Image Segmentation

Image semantic segmentation has been applied among various areas in computer vision and machine learning community. For example, techniques, like autonomous driving, indoor navigation, and virtual or augmented reality, all rely on image semantic segmentation mechanisms [20]. Given an image and objects, the evolution of object recognition in a coarse-to-fine manner can be described as: image classification, object detection/localization, semantic segmentation, and finally instance segmentation.

2.2.1 Semantic Segmentation

The fully-supervised methods like CNN for image classification task [21] leads to multiple networks for pixel-level labelling problems like semantic segmentation. The main advantage of these networks lies in learning feature representations for a specific problem in an end-to-end fashion.

Firstly, the Fully Convolutional Network (FCN) [16] is one of the variants. The high level idea of FCN is to transform a classification-driven CNN to pixel-wise prediction by replacing fully-connected layers with convolutional layers, as well as introducing deconvolutional layers to upsample the feature maps. FCN shows how CNNs are trained in an end-to-end manner for semantic segmentation task, resulting in a pixel-wise prediction regardless the arbitrary sizes of inputs [20]. FCN can achieve significant accuracy on some standard benchmarks (e.g., PASCAL VOC dataset), and it also preserves the computational efficiency during inference phase.

Secondly, Ronneberger et al. [22] proposed a network, U-net, and training strategy that strongly utilize data augmentation for biomedical image segmentation. In particular, the architecture for both capturing context and enabling precise localization. As a result, the U-net architecture performs well on many biomedical segmentation applications.

2.2.2 Instance Segmentation

Most approaches for instance segmentation are based on segment proposals [8] due to the development of R-CNN. For example, Hariharan et al. [23] introduced a novel architecture termed Simultaneous Detection and Segmentation (SDS), which is built based on R-CNN. This network aims at detecting all instances of a category and marking the pixels that belong to this category for each instance. Hariharan et al. [24] defined the hypercolumn at each pixel in an image as the vector of activations for all CNN units. These methods showed a significant improvement on the simultaneous detection and segmentation task. Dai et al. [25] proposed that utilizing the shape information is beneficial for object detection task by treating proposal segments like super-pixels as masks. Furthermore, they introduced a joint method to deal with objects and "stuff" (e.g., grass, sky, water). Pinheiro et al. [26] introduced a network called DeepMask, in which the network learns to propose segment candidates, then these segment candidates are classified by Fast R-CNN detector. In summary, the methods discussed above share a common feature: the segmentation is carried out before object detection. Similarly, Dai et al. [27] presented a Multi-Task Network Cascades that predicts segment proposals from bounding box proposals, and then classification is perform afterwards.

In order to combine segment proposal system and object detection system, Li et al. [28] proposed an end-to-end design called Fully Convolutional Instance-aware Semantic (FCIS) to detect and segment object instances simultaneously in a fully convolutional way. Since the object

classes, bounding boxes, and masks are addressed simultaneously, this solution can achieve high efficiency. However, it shows deficiency in segmenting overlapping instances [8].

Another type of instance segmentation approach is to start with pixel-wise classification, then cut the pixels that belong to the same category/class into several instances. For example, [29] proposed a network called InstanceCut, which outputs both an instance-agnostic semantic segmentation and all instance-boundaries. Also, [30] introduced an end-to-end solution by combining watershed transform and deep learning to produce an energy map in which object instances are represented as energy basins. The Sequential Grouping Networks (SGN) proposed by Li et al. [31] contains a sequence of neural networks, each tackling with a sub-grouping segmentation problem.

2.2.3 Mask R-CNN

The Mask R-CNN method was proposed by He et al [8]. This method has an extra branch for predicting segmentation mask of objects based on the Faster R-CNN. The intuition of proposing Mask R-CNN is Mask R-CNN provides a general framework for object instance segmentation along with object detection. In other words, Mask R-CNN can detect any object of interest in an image, and can output a segmentation mask for each instance of each object of interest as well. According to [8], Mask R-CNN was also applied to several tasks other than object detection. For example, Mask R-CNN has been used to estimate human poses. In summary, the proposed Mask R-CNN method outperforms all current methods in the community in all three tasks of COCO Challenge: instance segmentation, object detection, and person keypoint detection [32]. Mask R-CNN is, therefore, the state-of-the-art detection method on PASCAL VOC, COCO, and ILSVRC.

2.3 Online Hard Example Mining (OHEM)

One of the challenges for object detection approaches (e.g., Faster R-CNN) is that the datasets contain a large number of easy examples and a small number of hard examples. This challenge motivates a few learning techniques that deal with hard examples. A common solution for this challenge is called Bootstrapping, or hard negative mining [7]. Hard negative mining has been widely applied in the object detection research domain.

There are several hard example selection approaches that select hard examples for training deep networks. For example, Simo-Serra et al. [33] introduced a method that independently selects hard positive and negative example via selecting a larger set of random examples according to their loss. Similarly, Wang et al. [34] proposed an approach that finds hard negative patches from a large set using triplet loss given a positive pair of patches. Focusing on CNN for image classification, Loshchilov et al. [35] proposed a selection method that investigates online selection of hard examples for mini-batch Stochastic Gradient Descent (SGD) methods where the selection is based on loss as well.

Instead of focusing on image classification, Shrivastava et al. [7] proposed the online hard example mining approach that focuses on online hard example selection strategy for region-based object detectors (i.e., Fast R-CNN object detector). Specifically, the online hard example mining approach works as follows. Firstly, the convolutional feature map is computed using the CNN for an input image at SGD iteration t. Secondly, instead of using a sampled mini-batch [36], the RoI network will do a forward pass using the feature map and all the input RoIs. Note that the RoI network only includes the RoI pooling layer, two fully-connected layers, and the loss computation for each RoI. More specifically, those hard examples will be selected based on the loss, where the input RoIs are sorted by loss. Thus, a number of examples (i.e., hard examples)

will be selected, at which the current network is worst. Consequently, most computation is shared between all RoIs through the convolutional feature maps in forward propagation process.

2.4 Transfer Learning

In practice, training an entire CNN from scratch requires a sufficiently large dataset. In contrast, it is easy to take advantage of pretraining a CNN on a relatively large dataset (e.g., ImageNet, which contains 1.2 million images with 1000 categories) [37]. There are two scenarios in Transfer Learning [38].

The first scenario uses CNN as fixed feature extractor. Specifically, given a CNN pretrained on ImageNet, we remove the last fully-connected layer, and then treat the rest as a fixed feature extractor, and only retrain the classifier. For example, if tackling with a binary classification problem (i.e., a new dataset with only 2 classes), we need to remove the classifier with 1000 category for ImageNet, but to retrain a binary classifier instead.

The second scenario aims to fine-tune the CNN. This strategy is to not only retrain the classifier for the new dataset, but also fine-tune the weights of the pretrained network by continuing the backpropagation.

2.5 Fruit Detection and Segmentation

DeepFruits [4] proposed by Sa et al. is a fruit detection system using deep CNNs. The system applied Faster R-CNN for the task of fruit detection by transfer learning. They proposed a multi-modal Faster R-CNN model to combine both the color (RGB) and Near-Infrared (NIR) information together. As a result, their approach achieves 0.838 F1-score for the detection of the sweet pepper dataset. Note that this dataset was collected by their team. There are 100 training

images and 22 test images. Furthermore, this model is retrained on other datasets for the detection of seven fruits. Besides, Bargoti et al. [3] adopted Faster R-CNN on the task of fruit detection in orchards, including mangoes, almonds, and apples. Their study leads to the best yet detection performance in the author's line of prior work.

For tomato detection and segmentation, Yamamoto et al. [39] presented an image-processing method to detect tomato fruits in different growth stages, including mature, immature, and young fruits. Specifically, to start with, a pixel-based segmentation is performed to segment the pixels into different classes including fruits, leaves, stems, and backgrounds. Secondly, a blob-based segmentation is conducted to remove the misclassifications generated from the first step. Thirdly, they used K-means clustering [40] to detect each fruit in a fruit cluster. In summary, the results of fruit detection showed that the developed method achieved accurate detection performance, while detection of young fruits is very difficult due to their small size and the similarity of their appearances with that of stems.

CHAPTER 3

NETWORK ARCHITECTURES

Inspired by the Online Hard Example Mining algorithm for training Fast R-CNN object detector, I adopted the OHEM algorithm for both Faster R-CNN detector and Mask R-CNN architecture. They are named as Faster R-CNN with OHEM and Mask R-CNN with OHEM respectively. In this chapter, at the beginning I will introduce the details of Faster R-CNN with OHEM, which include the architecture of Faster R-CNN and Faster R-CNN with OHEM, and the training strategies for each approach as well. Then the details of Mask R-CNN with OHEM will be given.

3.1 Faster R-CNN with OHEM

3.1.1 Fast R-CNN

A Fast R-CNN network [15] takes a single image as input and a set of object proposals. Firstly, the network feeds the whole input image with several convolutional layers and max pooling layers. As a result, a convolutional feature map is generated. Secondly, a fixed-length feature vector is extracted from the feature map by a region of interest (RoI) pooling layer, for each object proposal. Thirdly, each feature vector is fed into a sequence of fully connected layers, producing two sibling output layers: classification layer and bounding box regression layer. On one hand, the classification layer produces Softmax probability that estimates over all object classes and a “background” class. On the other hand, the bounding box regression layer outputs

four real-valued numbers for each object class, annotating refined bounding-box positions for one class.

VGG-16 network, which was proposed by Simonyan et al. [41], is used as base network. It contains 13 shareable convolutional layers. Their major contribution lies in a networks of increasing depth using an architecture with a 3 by 3 convolution filters, which shows that a significant improvement through pushing the depth to 16 weight layers.

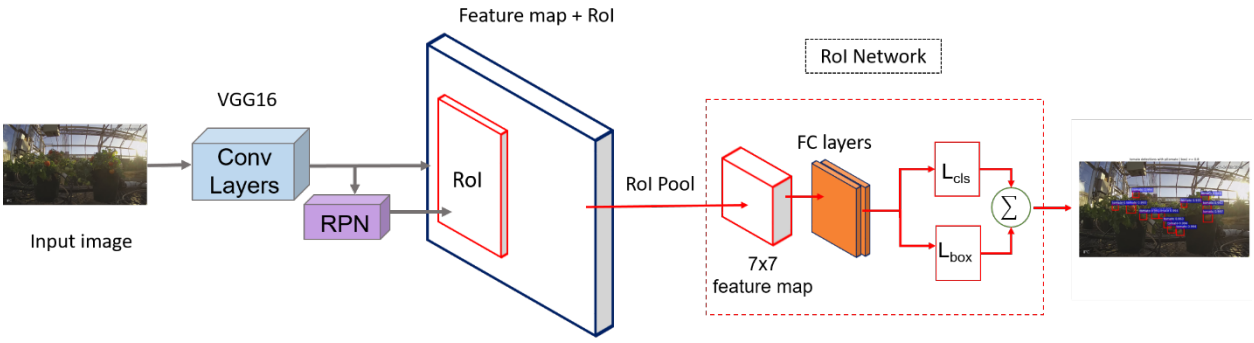

Figure 1. Faster R-CNN architecture. The backbone network is VGG16 net [41]. The area annotated with red dashed box is the Region of Interest (RoI) network.

3.1.2 Faster R-CNN

Since Faster R-CNN is the state-of-the-art framework for object detection and performs the best trade-off between accuracy and speed, I deployed Faster R-CNN framework for fruit detection task as a baseline.

Figure 1 shows the architecture of Faster R-CNN. The network takes an image as input, the shared convolutional layers are used to extract convolutional features, resulting in a feature map. The RoIs are generated from the Region Proposal Network. The RoIs are projected onto the feature map. Then the RoI Pooling layer pools the RoIs into a fixed-size feature map, followed by two fully-connected layers.

Specifically, in order to propose region proposals, a small network is slid over the convolutional feature map output, which is generated by the last shared convolutional layer. The input of this small network is an n × n spatial window of the convolutional feature map. In my case, n = 3 is used as spatial window. Each sliding window is mapped to a feature with lower dimension (e.g., 512-d for VGG). Then this feature map is fed into two fully-connected layers. The fully-connected layers are shared over all spatial locations.

We need to predict multiple region proposals at each location. For each location, the number of maximum possible proposals is denoted as k. Since I need to donate the four coordinates of the bounding boxes and two classification scores, the regression layer has 4 k outputs (i.e., the coordinates of k bounding boxes), and the classification layer has 2k scores (i.e., estimated probability of object or not object for each proposal). An anchor is associated with a scale and aspect ratio, centering at each sliding window. In Ren's work [11], they used three scales with box areas of 128^2, 256^2, and 512^2 pixels, and three aspect ratios: 1:1, 1:2, and 2:1. Consequently, there are k = 9 anchors at each sliding position. In the experiments, the same scales and aspect ratio are used for anchors.

Some proposals from RPN are highly overlapping with each other. In order to reduce these redundant and correlated regions, [7] used a standard non-maximum suppression (NMS) approach from [15]. Specifically, NMS selects the RoI with the highest loss iteratively given a list of RoIs and their corresponding losses, and then removes all RoIs with lower loss, in which having the high overlap with the selected region.

3.1.3 Training

Faster R-CNN [1] adopted a Four-step Alternating Training algorithm to learn shared features by optimization. Specifically, they first train the RPN. The RPN is initialized with a

model, pre-trained on ImageNet, thus is fine-tuned for the region proposal task in an end-to-end fashion. Secondly, they train the Fast R-CNN detection network using the region proposals that are generated by the RPN from the first step. This Fast R-CNN detection network is also initialized by the model, which was pre-trained on ImageNet. Now the two networks have not shared convolutional features yet. In the third step, the Fast R-CNN detector network is used to initialize RPN training, however, the shared convolutional layers are fixed and only fine-tune the layers that are identical to RPN. Consequently, both networks share convolutional layers at this point. In the fourth step, the shared convolutional layers remain fixed, then only fine-tune the layers that are unique to Fast R-CNN. As a result, the two networks share the same convolutional layers, producing a unified single network.

I applied the Four-step Alternating Training strategy as mentioned above to train the Faster R-CNN. The architecture for backbone network is identical to VGG-16-net. The VGG-16 network is composed of 13 convolutional layers followed by two fully-connected layers. The original Faster R-CNN was fine-tuned using the PASCAL VOC dataset, which was initialized by the pre-trained ImageNet dataset. I fine-tune the Faster R-CNN using our own dataset.

3.1.4 Faster R-CNN with OHEM

In [7], the OHEM algorithm was proposed for training Fast R-CNN object detector. Inspired by the OHEM algorithm for training Fast R-CNN object detector, I adopted the OHEM algorithm for training Faster R-CNN detector.

Figure 2 shows the architecture of the proposed Faster R-CNN with OHEM. On one hand, the Region Proposal Network remained the same as in Faster R-CNN [1]. On the other hand, both standard RoI network and Read only RoI network are applied.

More specifically, other than regular RoI network in Fast R-CNN, the other RoI network, which is called Read only RoI network, is introduced. Note that the standard RoI network allocates memory for both forward and backward passes. However, oppositely, the memory is allocated only for forward pass in the Read only RoI network. Then the Read only RoI network does a forward pass, computing loss for all input RoIs, given the convolutional feature map for each SGD iteration. Furthermore, a hard RoI sampling module is applied to select hard examples. Note that all RoIs from all images are annotated as R. The effective batch size for the Read only RoI network is |R|. However, for the standard RoI network, the batch size remains the same as in Fast R-CNN implementation.

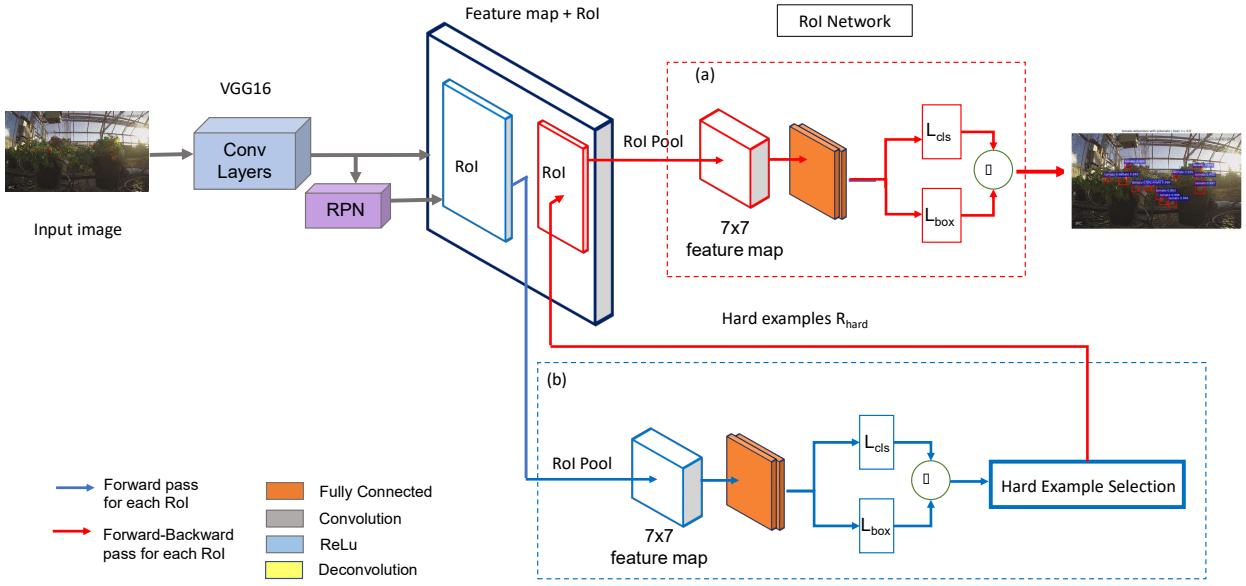

Figure 2. The architecture of Faster R-CNN with OHEM. The backbone network is VGG16 net. The area annotated with red dashed box is the standard RoI network, while the one with blue dashed box is the Read only RoI network.

I implemented the Faster R-CNN with OHEM framework that illustrated above using the Caffe [42] deep learning framework. For forward-backward passes, N = 1 and B = 128 are used. In other words, for each mini-batch, one image is first sampled from the dataset, and then B/N RoIs (i.e., 128 RoIs) are sampled from each image.

3.2 Mask R-CNN with OHEM

3.2.1 Mask R-CNN framework

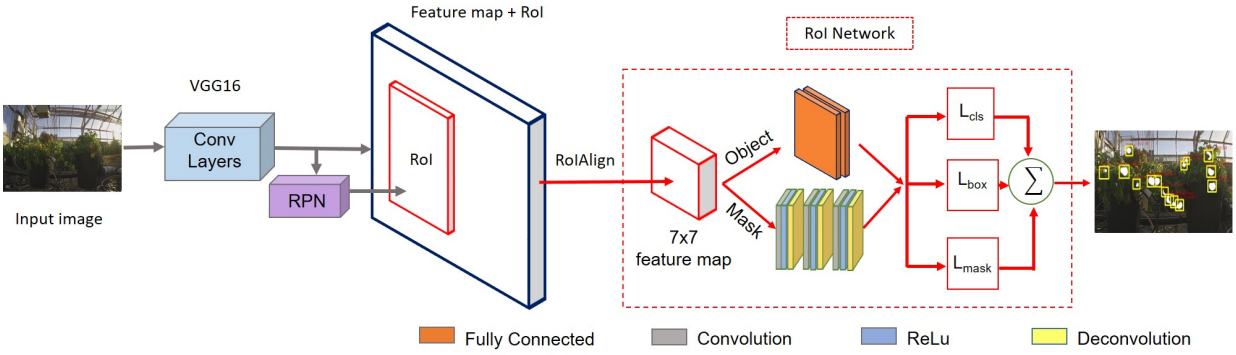

Figure 3. Mask R-CNN architecture. The backbone network is VGG16 net. The area annotated with red dashed box is the RoI network. There are two braches in the RoI network: one is for object detection, the other is for mask prediction.

Instance segmentation requires not only the precise detection of the objects, but also the segmentation of each instance. Therefore, it is a challenging task. More specifically, the goal of instance segmentation is to localize each object instance in the image, and simultaneously to classify each pixel into different categories.

Since our goal is to automatically detect tomatoes in the greenhouse, while measure the growth rates of the tomatoes, the Mask R-CNN framework for instance segmentation becomes one of the solutions to tackle with the task.

Mask R-CNN is an extension of Faster R-CNN framework. As discussed earlier, the outputs of Faster R-CNN for each candidate object are a class label and the bounding box. And Mask R-CNN adds a mask branch to the Faster R-CNN, which is the most distinct part from Faster R-CNN. Mask R-CNN uses the same two-stage procedure that is used in Faster R-CNN. Specifically, on one hand, the first training stage for Region Proposal Network in Mask R-CNN is identical to that in Faster R-CNN. On the other hand, in the second stage, Mask R-CNN predicts the classification label and bounding boxes, and also predicts a binary mask for each RoI.

Figure 3 shows the Mask R-CNN framework for object detection and instance segmentation. Mask R-CNN extends Faster R-CNN framework by adding a mask branch to Faster R-CNN for predicting an object mask in parallel with the existing branch for bounding box recognition. Similar to Faster R-CNN framework, the network takes an image as input. The shared convolutional layers are used to extract convolutional features, resulting in a feature map. The RoIs are generated from the Region Proposal Network. The RoIs are projected onto the feature map. Then the RoIAlign layer pools the RoIs into a fixed-size feature map. Two fully-connected layers are used for classification and bounding box regression. The mask branch consists a sequence of deconvolutional layer to upsample the feature map to a large scale. The details of RoIAlign layer and a sequence of deconvolutional layers will be given later.

During training, a multi-task loss function is defined for each sampled RoI as follows:

$$L = L_{cls} + L_{box} + L_{mask} \qquad (1)$$

Note that the classification loss L_{cls} and bounding box loss L_{box} are defined the same as that in Faster R-CNN. More specifically, the classification loss is the log loss function. They use smooth-L1 loss on the position of top-left of the box, and the logarithm of the height and width for bounding box regression. For mask branch, assume that we have K classes, and the resolution of each binary mask is m × m, then the output of the mask branch is Km^2-dimensional for each RoI. The mask loss is defined as the average binary cross-entropy loss. A Fully Convolutional Network (FCN) [16] is used to predict an m × m mask from each RoI, predicting a segmentation mask pixel-to-pixel.

In the original Mask R-CNN implementation [8], they extend two existing Faster R-CNN heads. One is ResNet C4 [43], the other is Feature Pyramid Network (FPN) [44]. For ResNet C4, features are extracted from the final convolutional layer of the 4-th stage in the original implementation of Faster R-CNN with ResNet. Therefore, it is called C4. For FPN, RoI features are extracted from various levels of the feature pyramid based on the scale in the implementation of Faster R-CNN with FPN backbone.

Specifically, the network architecture that is used is shown in Figure 3 for training Mask R-CNN instead of using the network architecture of Faster R-CNN with ResNet C4 and FPN in the original Mask R-CNN implementation. There are three key components in this Mask R-CNN: RoIAlign layer, A sequence of deconvolutional layers, and the end-to-end architecture.

3.2.2 RoIAlign layer

The first one is RoIAlign layer, which is identical to the one that is used in the original Mask R-CNN implementation. According to [8], this RoIAlign strategy plays an important and significant role in those segmentation tasks based on pixel level (e.g., image segmentation).

The RoIAlign layer, firstly introduced by [7], is to align the extracted features with the input. It tacks with my limitation that RoIPool layer owns. The RoIPool layer introduced in Faster R-CNN architecture is a standard pooling operation to pool the feature map into a smaller feature map (e.g., 7 × 7) from each Region of Interest. The purpose of replacing RoIAlign with RoIPool is to get rid of any quantization of the RoI boundaries or bins.

RoIAlign layer removes the harsh quantization RoI Pooling, properly aligning the extracted feature with the input. More specifically, RoIPool contains two steps of coordinates quantization: one is from the original input image into feature map (i.e., generated from the last convolutional layer), the other is from feature map into RoI feature. The RoIPool layer works by dividing the Region of Interest into a regular grid, and then performing max-pooling on the feature map in each grid cell. These quantization results in a huge loss of location precision. However, RoIAlign removes these two quantization, and manipulates coordinates on continuous domain, which increase the location accuracy significantly. As shown in Figure 4, instead of using the rounding operation, the RoI is divided into 2 × 2 sub-windows/bins. There are four sampling points in each bin. RoIAlign layer applies bilinear interpolation [45] to calculate the interpolated values of the input features at four regularly sampled locations in each RoI bin, then aggregates the result using max operation.

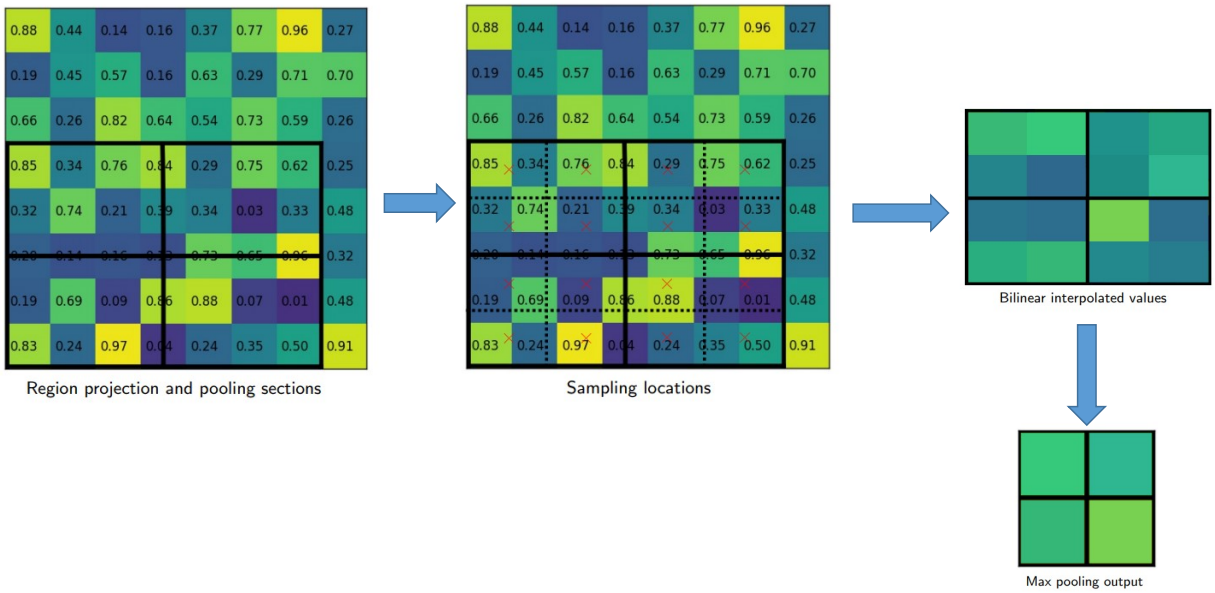

Figure 4. The details of RoIAlign layer (image source: Kaiming He)

3.2.3 Upsampling layers

The second key component is a sequence of deconvolutional layers. In the original Mask R-CNN implementation, the object segmentation mask size is fixed (e.g., 14 × 14 or 28 × 28). However, three deconvolutional layers are used to obtain segmentation mask with higher resolution.

Specifically, a 7 × 7 fixed size feature map is generated by the RoIAlign layer, then three deconvolutional layers are applied subsequently. Given an input feature map, annotated as S_i, the deconvolutional layer operation acts as the opposite way as convolutional layer operation, resulting in a larger feature map, annotated as S_o. And the relationship between S_i and S_o. is defined in the following equation:

$$S_0 = s\times(S_i\text{-}1) + S_f\text{-}2\times d \qquad (2)$$

where s is the stride step, d is padding, S_f is kernel size or filter size. These parameters for the three deconvolutional layers are shown in Table 2:

Table 2. The parameters for each deconvolutional layer in Mask R-CNN.

Deconvolutional layers	**Stride s**	**Padding d**	**Kernel size S_f**	**Input feature map S_i**	**Output feature map S_0**
Deconv 1	4	1	8	7 × 7	30 × 30
Deconv 2	4	1	8	30 × 30	122 × 122
Deconv 3	2	1	4	122 × 122	244 × 244

For instance, the output feature map given by the Deconv 1 layer is calculated as follows: $S_o = 4 \times (7 - 1) + 8 – 2 \times 1 = 30$, which means the output feature map size is 30 × 30. Similarly, the output feature map given by the Deconv 3 layer is computed as follows: $S_o = 2 \times (122 -1) + 4 – 2 \times 1 = 244$, resulting in a 244 × 244 output feature map. In summary, three deconvolutional layers are used to up-sample the feature map to the size of 244 × 244. More details are shown in Figure 5.

3.2.4 End-to-end architecture

Figure 3 show the architecture of this end-to-end network. The network consists of two branches, one is detection branch for object detection, the other is a mask branch for instance segmentation. Note that the architecture for object detection branch is identical to the one in Faster R-CNN. Compare to original Mask R-CNN implementation, VGG16 network instead of ResNet C4 or FPN backbone is used to extract the features given an input image. As in Faster R-

CNN, the RPN is used to generate Region of Interests (RoIs) or candidate bounding boxes by sharing the VGG16 convolutional backbone. On one hand, for each RoI, the image feature map from the last convolutional layer of VGG16 (i.e., conv5_3) is pooled into a 7 × 7 feature map by the RoIAlign layer. On the other hand, the mask branch up-samples the 7 × 7 feature map to 244 × 244 feature map, which has higher resolution. Furthermore, a Softmax layer is used to assign each pixel in the output feature map to its corresponding class. Finally, this network is trained end-to-end.

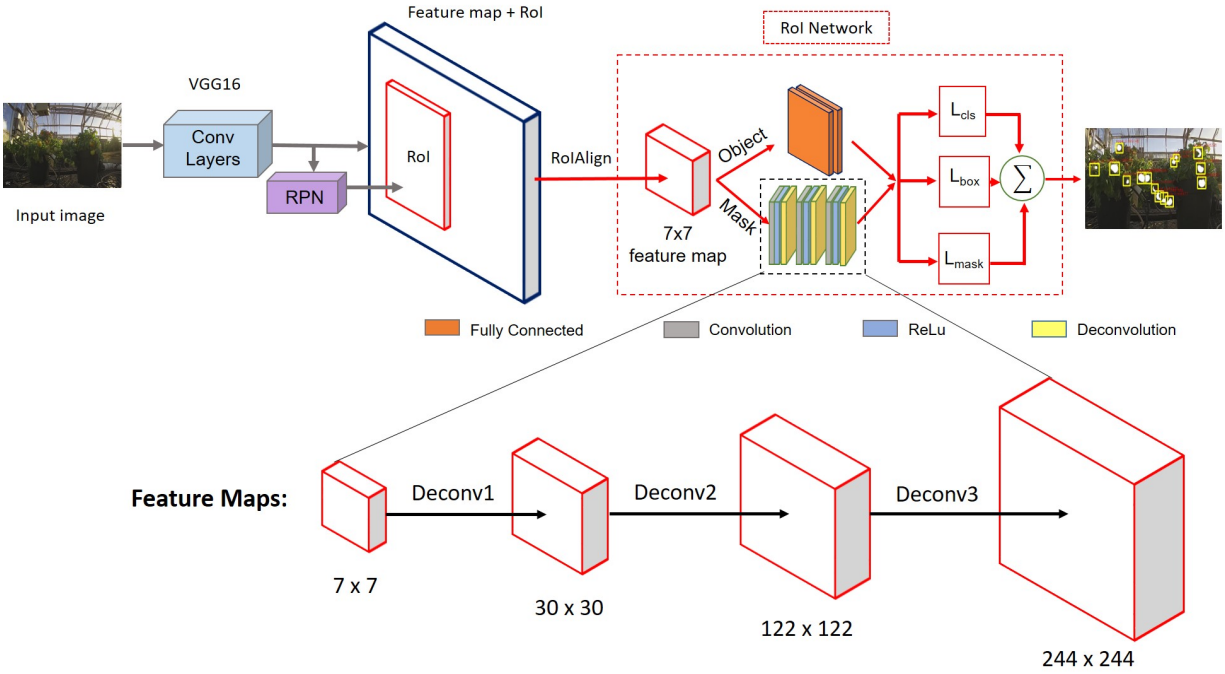

Figure 5. A sequence of deconvolutional layers for up-sampling the feature map in Mask R-CNN.

3.2.5 Network architecture of Mask R-CNN with OHEM

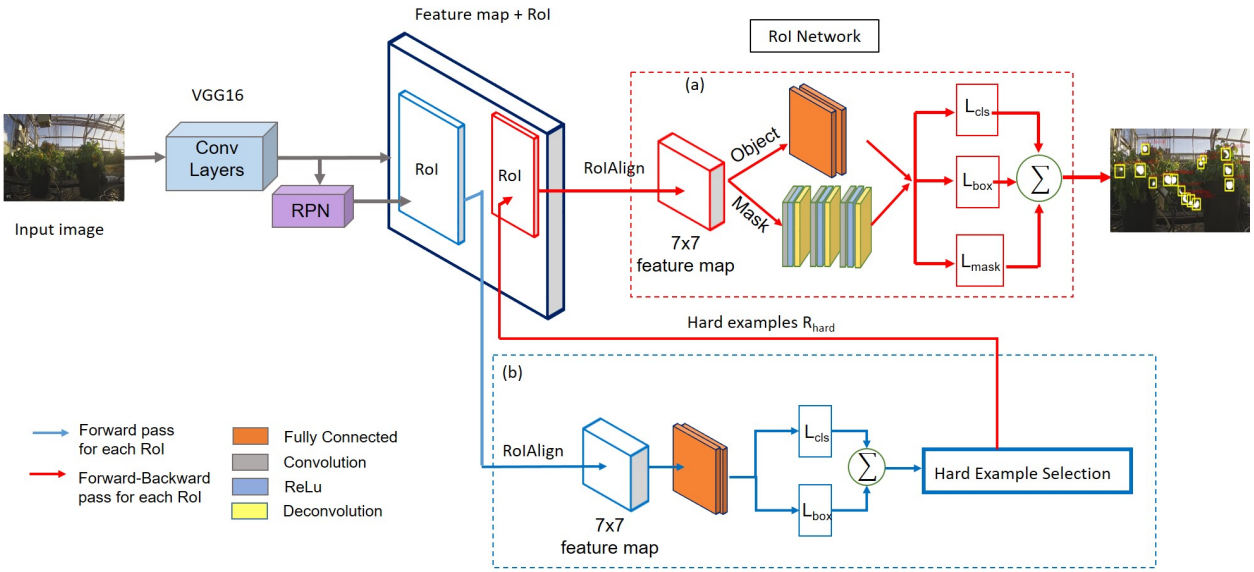

Figure 6. The architecture of Mask R-CNN with OHEM. The backbone network is VGG16 net. The area annotated with red dashed box is the standard RoI network. The one with blue dashed box is the Read only RoI network.

Specifically, I adopted the design of the OHEM module in Shrivastava et al's work [7] for Mask R-CNN. In the Mask R-CNN with OHEM architecture, there are two RoI networks, one is the standard RoI network in Faster R-CNN, the other is called the Read only RoI network. The details are given below. In each SGD iteration, the RPN generates RoIs given an image, and the convolutional network generates a convolutional feature map as shown in Figure 6. The Read only RoI network, which includes the Hard RoI module, computes the loss over all input RoIs (i.e., |R|). Then the RoIs will be sorted according to the loss. Therefore, the hard examples (i.e., the ones with higher loss) are selected as input to the standard RoI network. The forward pass as well as backward pass are performed on this standard RoI network only for those hard examples

(i.e., $|R_{hard}|$). Note that the effective batch size for the Read only RoI network is the number of the hard examples selected $|R_{hard}|$, and the standard RoI network adopts the mini-batch strategy as in Faster R-CNN.

3.2.6 Implementation of Mask R-CNN with OHEM

Since I have applied the OHEM on Faster R-CNN, I observe that the OHEM helps to improve the detection results, especially in increasing the Recall value. Therefore, it's intuitive and straightforward to come up with the idea that we can combine OHEM with Mask R-CNN as well. On one hand, we assume that the hard example mining strategy at feature level can improve the detection performance. On the other hand, by adding the mask branch to the object detector can improve the detection result as well because the predicted mask can provide valuable hints on some features like shape information.

As mentioned earlier, the Four-step training strategy is adopted for training the Fast R-CNN. However, the end-to-end training strategy is used for training the Mask R-CNN. Therefore, I trained the Mask R-CNN with OHEM using the end-to-end training scheme, which is identical to the training strategy of Mask R-CNN.

CHAPTER 4

EXPERIMENTS

In this Chapter, I qualitatively and quantitatively evaluate each network as follows: (1) the results of Faster R-CNN for different types of fruits; (2) the results of Faster R-CNN with OHEM; (3) the results of Mask R-CNN and Mask R-CNN with OHEM for tomato dataset only; (4) a comparison of the performance for the approaches mentioned above. At the end, the applications of tomato detection and segmentation will be addressed.

4.1 Datasets

4.1.1 Apple and Mango dataset

Table 3 shows the number of training images and test images. Note that the Apple and Mango datasets are obtained from [19]. One can easily observe that the number of images is relatively small because of the limited image annotation datasets from [20]. In order to compare different approaches in a fair fashion, we used the same training and testing images for all proposed approaches, and the experimental results are shown in the Chapter 4.

Table 3. The number of images used for both training and testing for different fruits.

Name of Fruits	Training(# of images)	Test (# of images)	Total (# of images)
Apple	51	10	61
Mango	113	23	136

4.1.2 Tomato dataset

Table 4 shows the number of training images and test images for our tomato dataset. For tomato detection and segmentation, we obtained all the images from two security cameras. Both cameras are placed by the side of two tomato plants. Firstly, we tried to put the cameras right above the tomato plants to capture more tomato plants in a single image, but it turned out that we can obtain better observations of tomatoes, which is more beneficial for our tomato detection task. The cameras have both day and night mode, however, we did not collect those images at night. In other words, the images in our tomato set used in all experiments are color (RGB) images. Specifically, we collected four images per day at 9:00 am, 11:00 am, 2:00 pm, and 5:00 pm respectively. Moreover, the collection period started from middle December to the end of January. The purpose to collect images from different time points in a single date is to monitor the tomatoes under different light conditions. For example, the light conditions at 11:00 am and 2:00 pm are generally much stronger than that at 9:00 am and 5:00 pm.

Note that there are several datasets for tomato detection and segmentation. The difference among those datasets are given below. On one hand, two datasets are used for tomato detection, the images in each dataset are obtained from Camera 1 and Camera 2 respectively. On the other hand, for tomato segmentation, I have generated the mask as groundtruth for tomatoes from the first camera, only since it is significantly time-consuming to generate the groundtruth mask for any kind of supervised segmentation task. The first dataset for tomato segmentation contains 128 images in total from the first camera, the size of these images is 1280 × 720. The second dataset for tomato segmentation contains the images that are cropped into size of 900 × 500 from the images in the first dataset. The purpose doing this is to focus much more on the tomato plants.

Table 4. The number of images used for both training and testing for tomato.

Name of Fruits	Task	Image Size	Training (# of images)	Test (# of images)	Total (# of images)
Tomato (Camera 1)	Detection	1280 × 720	112	28	140
Tomato (Camera 2)	Detection	1280 × 720	102	26	128
Tomato (Camera 1 + Camera 2)	Detection	1280 × 720	214	54	268
Tomato (Camera 1)	Detection + Segmentation	1280 × 720	100	28	128
Tomato (Camera 1)	Detection + Segmentation	900 × 500	100	28	128

Figure 7. Four images at different time slots on the same day from Camera 1 (i.e., 01/04/2018).

Figure 8. Four images at different time slots on the same day from Camera 2 (i.e., 01/04/2018).

Figure 7 and 8 show a comparison of images in different time slots within a single day from Camera 1 and Camera 2 respectively. Furthermore, for ground truth labelling, we used the same strategy for all fruit datasets including tomato dataset. A Matlab script is used for drawing the bounding box for each object of interest. Note that we labeled all tomatoes as ground truth, even though they appear tiny in the image.

4.2 Evaluation Metrics

In this section, evaluation metrics for both object detection and instance segmentation are given as follow. Note that the evaluation metrics including Precision, Recall, and F1-score are used to measure the performance of Faster R-CNN and Faster R-CNN with OHEM. However,

other than these three metrics for object detection, Dice Score is used to evaluate the segmentation performance of Mask R-CNN and Mask R-CNN with OHEM.

4.2.1 Evaluation measures for object detection

In my work, I applied precision, recall, and along with the corresponding F1 score as evaluation metric to evaluate the performance of each approach for fruit detection. More specifically, an object is considered as detected when the Intersection of Union (IoU) between the predicted bounding boxes and ground truth bounding boxes is greater than a certain threshold (e.g., 0.7). Note that the threshold affects the performance evaluations Although the threshold affects the performance evaluations. Theoretically, the smaller the threshold is, the higher the F1 score will be). The Precision, Recall, and F1 score are computed as following given the IoU threshold:

$$\text{Precision}=\frac{\text{T}_\text{p}}{\text{T}_\text{p}+\text{F}_\text{p}} \qquad (3)$$

$$\text{Recall} = \frac{T_p}{T_p+T_n} \qquad (4)$$

$$\text{F1} = \frac{2\times Precision\times Recall}{Precision+Recall} \qquad (5)$$

where T_p: the number of true positives (correct detections), F_p : the number of false positives (false detection), F_n: the number of false negatives (miss) and T_n is the number of true negatives (correct rejection).

4.2.2 Evaluation measures for instance segmentation

I adopted one of the evaluation measures called Dice Score [46] of binary segmentations:

$$\text{Dice}(\%)=\frac{2\left|\text{P}^{\text{gt}}\cap\text{P}^{\text{pd}}\right|}{\left|\text{P}^{\text{gt}}\right|+\left|\text{P}^{\text{pd}}\right|} \qquad (6)$$

where P^{gt} is ground truth mask, and P^{pd} is the predicted result or algorithmic result. This measure evaluates the degree of overlapping between predicted mask and ground truth mask.

4.3 Experimental Results

4.3.1 Experimental results on Apple and Mango Dataset

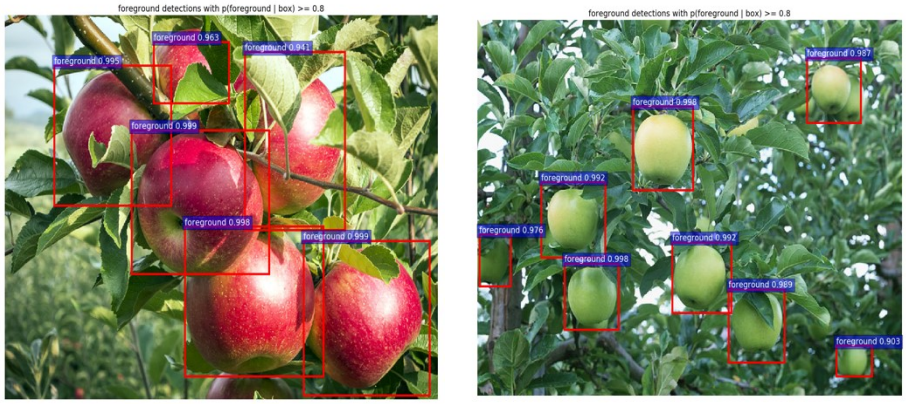

Figure 9. Test images from Apple dataset and detection results using Faster R-CNN.

Figure 9 shows two test images from Apple dataset on Faster R-CNN without any hard example mining strategies. Note that the red rectangle box denotes the object that is predicted to be detected, along with the corresponding confidence score. For example, for the test image on the left, all detected bounding boxes contain the object of interest (i.e., apple). Therefore, the number of true positives is 6. Moreover, there is no false negative in this image.

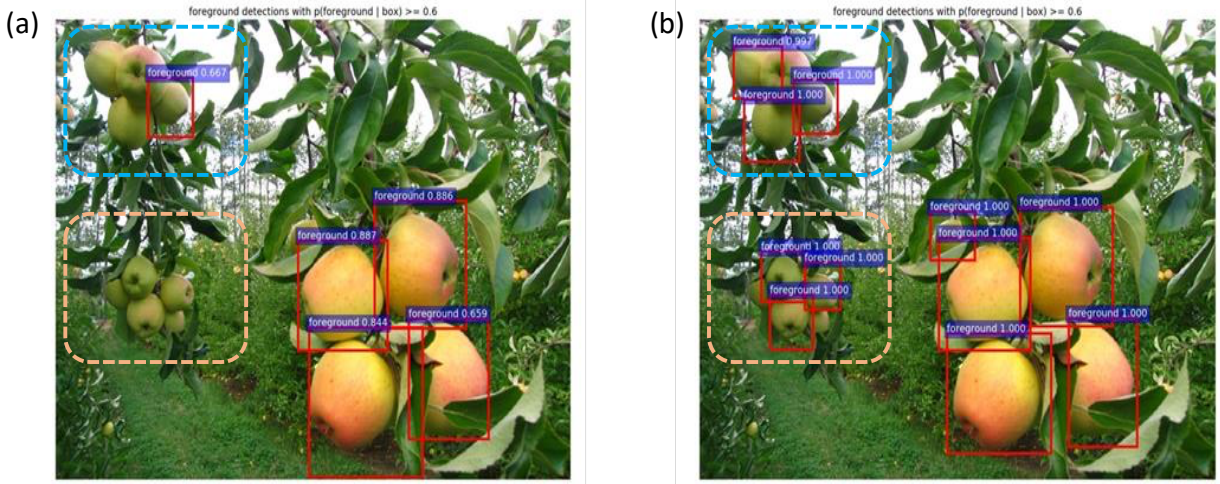

Figure 10. (a): Result on Faster R-CNN with hard example mining on image level; (b): Result on Faster R-CNN with OHEM algorithm. Note that the dashed regions in blue and yellow show two differences between these two detected results, respectively.

Figure 10 shows a comparison of results between the network trained on Faster R-CNN with hard example mining and Faster R-CNN with OHEM algorithm. Note that the approach of Faster R-CNN with hard example mining means after inference, the images with hard examples (i.e., false positives and false negatives) are selected manually, then these selected images are used to retrain the network.

We can observe that the region annotated as blue shows the differences between these two approaches. For the image on the left, the network is not able to detect most of the apples within this region, while for the image on the right, the network can detect three apples out of four in this region. Similarly, within the region annotated as yellow, none of the five apples are detected in the image on the left, while three apples out of five are detected in the image on the

right. To conclude, the Faster R-CNN with OHEM approach performs better on this single test image. The quantitative results will be shown later.

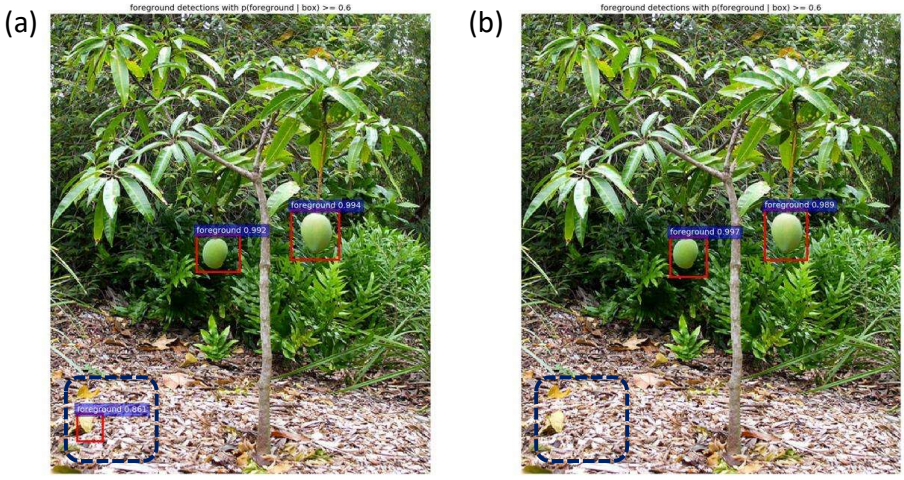

Figure 11. (a): Result on Faster R-CNN; (b): Result on Faster R-CNN with OHEM. Note that the dashed region in blue shows the difference between these two detected results.

Figure 11 shows a comparison of results between the network trained on Faster R-CNN and Faster R-CNN with OHEM algorithm. Observer that in (a), the predicted bounding box on the bottom left is a false positive example, which is a false detection of mango.

Table 5: Performance evaluation of different approaches on Apple dataset.

Approach	Precision	Recall	F1-score
Faster R-CNN	0.750 (±0.14)	0.966 (±0.125)	0.844 (±0.119)
Faster R-CNN (Hard)	0.803 (±0.123)	0.929 (±0.076)	0.861 (±0.113)
Faster R-CNN with OHEM	**0.870 (±0.087)**	**0.959 (±0.081)**	**0.912 (±0.081)**

Table 6. Performance evaluation of different approaches on Mango dataset.

Approach	Precision	Recall	F1-score
Faster R-CNN	0.783 (±0.128)	0.864 (±0.082)	0.822 (±0.104)
Faster R-CNN (Hard)	0.791 (±0.118)	0.872 (±0.010)	0.837 (±0.079)
Faster R-CNN with OHEM	**0.809 (±0.073)**	**0.884 (±0.093)**	**0.845 (±0.098)**

Table 5 and 6 show the quantitative results for Faster R-CNN, Faster R-CNN (Hard), and Faster R-CNN with OHEM. Note that the Faster R-CNN (Hard) means the approach that manually selects the images with hard example (false positives and false negatives), then retrain the network using these hard examples images only. Also note that all evaluation metrics including Precision, Recall, and F1 score are computed based on the IoU threshold at 0.7.

Here are some observations: as shown on this table, applying the hard example mining on image level, in other words, by heuristically choosing images that contain hard examples to train the network again can increase the Precision value and decrease the Recall value, ending up with slight increase of F1 score for this Apple dataset. Furthermore, instead of selecting hard examples on image level, adopting online hard example mining algorithm for training Faster R-CNN detector increases the Precision value by 10% compared to solely training Faster R-CNN for the same dataset. However, the Recall value for Faster R-CNN with OHEM approach slightly drops to 0.959. Nevertheless, Faster R-CNN with OHEM outperforms other two methods on F1 score.

4.3.2 Experimental results on Tomato Dataset

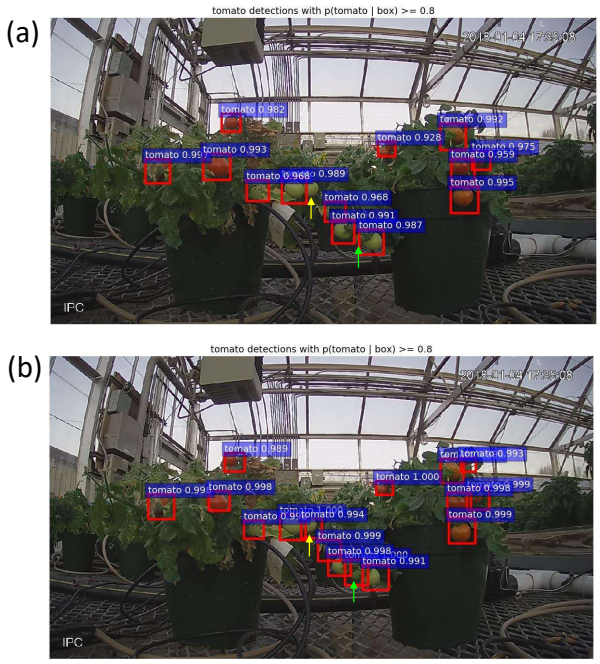

Figure 12. (a): Detection result on Faster R-CNN; (b) Detection result on Faster R-CNN with OHEM. Note that the yellow and green arrows indicate two differences between these two detected results. The highlighted tomatoes are not detected in (a).

Figure 12 shows the detection result of Faster R-CNN and Faster R-CNN with OHEM respectively on our tomato dataset (from Camera 1). We can observe that Faster R-CNN with OHEM can detect some instances that are not detected by Faster R-CNN framework. The details are highlighted as yellow and green, respectively.

I applied both Faster R-CNN and Faster R-CNN with OHEM on our tomato dataset. Hereafter, I call the dataset that contains images collected from Camera 1, the dataset that contains images from Camera 2, and the dataset contains the combination of images from both

Camera 1 and Camera 2. In this section, I will show the experimental results considering Camera 1 and Camera 2 separately, and the combination set as well.

Table 7. Test results on different approaches (training images are from Camera 1 only, 28 test images)

Approach	Precision	Recall	F1-score
Faster R-CNN	**0.956 (±0.086)**	0.715 (±0.138)	0.818 (±0.133)
Faster R-CNN with OHEM	0.923 (±0.098)	**0.784 (±0.127)**	**0.848 (±0.121)**

Table 8. Test results on different approaches (training images are from Camera 2 only, 26 test images)

Approach	Precision	Recall	F1-score
Faster R-CNN	**0.968 (±0.044)**	0.878 (±0.084)	0.914 (±0.073)
Faster R-CNN with OHEM	0.930 (±0.067)	**0.916 (±0.098)**	**0.919 (±0.082)**

Table 9. Test results on different approaches (training images are from two cameras, 54 test images)

Approach	Precision	Recall	F1-score
Faster R-CNN	**0.942 (±0.086)**	0.847 (±0.135)	0.876 (±0.112)
Faster R-CNN with OHEM	0.915 (±0.078)	**0.879 (±0.114)**	**0.890 (±0.105)**

Table 7 shows the experimental results of Faster R-CNN and Faster R-CNN with OHEM for the dataset Camera 1. Table 8 shows the experimental results of Faster R-CNN and Faster R-CNN with OHEM for the dataset Camera 2. Table 9 shows the experimental results of Faster R-CNN and Faster R-CNN with OHEM for the combination dataset.

In summary, when training with dataset Camera 1 only using Faster R-CNN with OHEM, the Recall value increases by 7%, and F1-score increases by 3%, compared to Faster R-CNN. On the other hand, when training with dataset Camera 2 only, the Recall value increases by 4%, along with a slight increase in F1-score. To conclude, augmenting Faster R-CNN framework with OHEM enables the decrease of Precision value and the increase of Recall and F1-score.

4.3.3 Experimental results on Mask R-CNN and Mask R-CNN with OHEM

4.3.3.1 Data preparation and ground truth generation

The input of Mask R-CNN for tomato segmentation requires both the bounding box annotation and the ground truth for each instance in a single image. Similar to tomato detection, dataset is the same as the one used in tomato detection, however, I am only using the images from Camera 1. In total, there are 100 training images, and 28 images for testing.

For ground truth mask generation, since the network requires one mask image for each instance, it's time-consuming and requires a lot of human labor. Here I list two ways generating the ground truth mask. One is utilizing an application called DeepSegments (https://github.com/andrewcking/deep-segments). DeepSegments is a tool for generating ground-truth segmentations for use in deep learning segmentation models. It provides a simple method for researchers to quickly segment their datasets. Users can choose from two different segmentation methods, SLIC or graph cuts. The program will make label suggestions by propagating user-given labels to unlabeled portions. Label suggestions can be given by an

unsupervised k-means algorithm or a user-supplied pre-trained model. The other tool that is used is an Image Processing tool in Matlab called Image Segmenter. Specifically, by clicking the Draw Freehand function, we can easily draw a mask for the tomatoes and save it into a binary image.

However, since the ground truth generated by DeepSegments applies either the SLIC or graph cuts method, the results are not accurate enough, especially when the boundaries of the tomatoes share the similar color intensity with the leaves near by the tomatoes. That is one of the drawbacks of utilizing SLIC. Therefore, I chose the Image Segmenter tool in Matlab as a ground truth generation method. Note that I also need to generate the ground truth mask for test set in order to do experimental evaluation.

4.3.3.2 Training and inference of Mask R-CNN

More specifically, I trained the network end to end using Adam [47] optimizer with 0.9 momentum, 0.999 momentum2, and 10^{-8} delta. Adam is a gradient-based optimization method like SGD [48]. The network was trained on a Titan X GPU with 12 GB for 20k iterations. The learning rate is fixed to 0.001. Similar to He et al.'s work, 15 anchors are used in the RPN including 5 scales and 3 aspect ratios. To compute the multi-task loss, which is mentioned earlier, the top 2000 RoIs are chosen from the RPN. Note that an RoI will be considered as positive when the Intersection of Union (IoU) between the groundtruth box and the proposed RoI is larger or equal to 0.5. And an RoI is considered as negative otherwise. I chose the mini-batch size as 40. Note that 40 is the maximum value due to the memory limitation on the GPU.

On the other hand, in testing phase, I choose the top 1000 RoIs which are generated by the RPN, and the object detector is performed on these RoIs, followed by a NMS. I set a classification threshold to be 0.8 so that the predicted bounding boxes are selected from the

outputs of the detection branch when the classification score is larger or equal to 0.8. Then the mask branch takes these detected objects as inputs to generate binary mask for each instance.

4.3.3.3 Training and inference of Mask R-CNN with OHEM

Similar to the training on our Mask R-CNN, I trained the Mask R-CNN with OHEM in an end-to-end manner using Adam optimizer in Caffe with 0.9 momentum, 0.999 momentum2, and 10^{-8} delta. The same setting is used as in training Mask R-CNN: the network was trained on a Titan X GPU with 12 GB, but for 25k iterations instead. Note that I choose 25 as mini-batch size since 25 is the maximum value available due to the memory limitation. The learning rate is fixed to 0.001. There are 15 anchors in the RPN including 5 scales and 3 aspect ratios.

During testing, the classification threshold is set to be 0.8, which is identical to the one in Mask R-CNN.

Table 10. An overview of the parameters of different approaches for training.

Approach	Optimizer	Momentum	Momentum2	Delta	Batch-size	Learning Rate
Faster R-CNN	Adam	0.9	0.999	10^{-8}	128	0.001
Faster R-CNN with OHEM	Adam	0.9	0.999	10^{-8}	128	0.001
Mask R-CNN	Adam	0.9	0.999	10^{-8}	40	0.001
Mask R-CNN with OHEM	Adam	0.9	0.999	10^{-8}	25	0.001

Table 10 shows an overview of the parameters used in different approaches during training phase. Note that the Adam optimizer is used in all approaches with the same momentum, momentum 2, and delta value.

4.3.3.4 Experimental results

In this section, a thorough discussion of the qualitative and quantitative results of tomato detection and segmentation will be given.

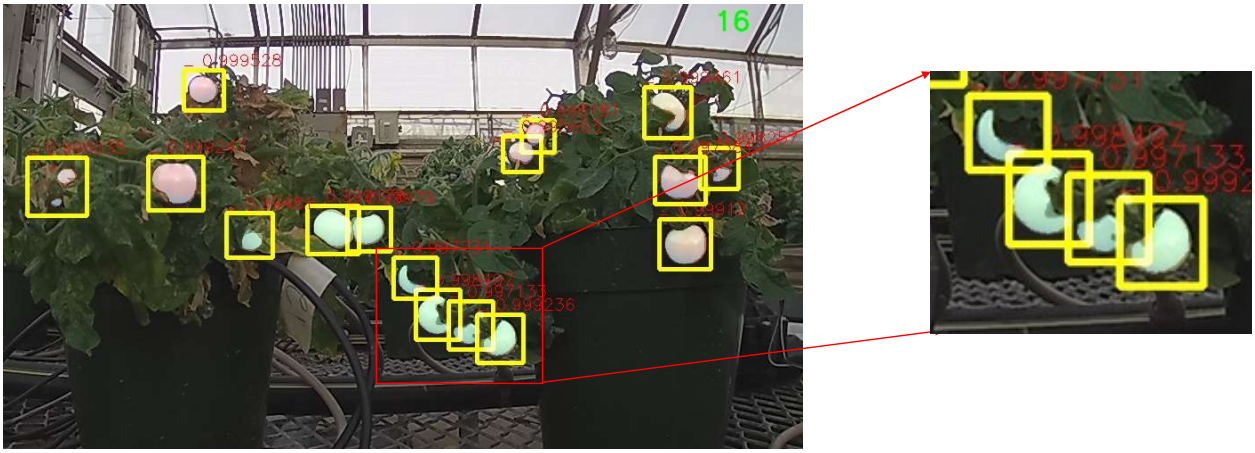

Figure 13. A qualitative result of Mask R-CNN with OHEM. Note that the transparent blue pixels demonstrate the predicted mask, while the transparently white pixels show the ground truth mask.

Figure 13 shows a qualitative result of Mask R-CNN with OHEM. This network predicts the locations of the tomato and the segmentation masks simultaneously. Specifically, the yellow bounding boxes illustrates the locations of the tomatoes, the transparent blue pixels demonstrate the predicted mask for each instance within the corresponding bounding box, and the transparently white pixels show the ground truth mask for each instance, which is annotated by

ourselves. Furthermore, the predicted number of bounding boxes (i.e., number of tomatoes) are shown on the top-right corner of the images.

Table 11. A comparison between different approaches on tomato detection and segmentation.

Network	Precision	Recall	F1-score	Dice Score
Faster R-CNN	0.936 (±0.085)	0.805 (±0.121)	0.859 (±0.104)	--
Faster R-CNN with OHEM	0.934 (±0.076)	0.863 (±0.116)	0.892 (±0.097)	--
Mask R-CNN	**0.991 (±0.032)**	0.917 (±0.113)	0.948 (±0.093)	0.806 (±0.020)
Mask R-CNN with OHEM	0.986 (±0.037)	**0.935 (±0.095)**	**0.957 (±0.079)**	**0.816 (±0.016)**

Table 11 shows a comparison between different approaches after correction for ground truth mask on both training and test sets. The ground truth masks on both training and test sets are not accurate enough since I had included the parts which are occluded by leaves when I generated the ground truth. Therefore, I decided to correct those instances on both training and test sets. As shown in the table, after correction, the Precision values for Faster R-CNN, Faster R-CNN with OHEM, and Mask R-CNN drop slightly, while Recall values for all approaches increase. Overall, the F1-score for all approaches increases, compared to the results before correction. Furthermore, the dice score for both Mask R-CNN and Mask R-CNN with OHEM increases as well.

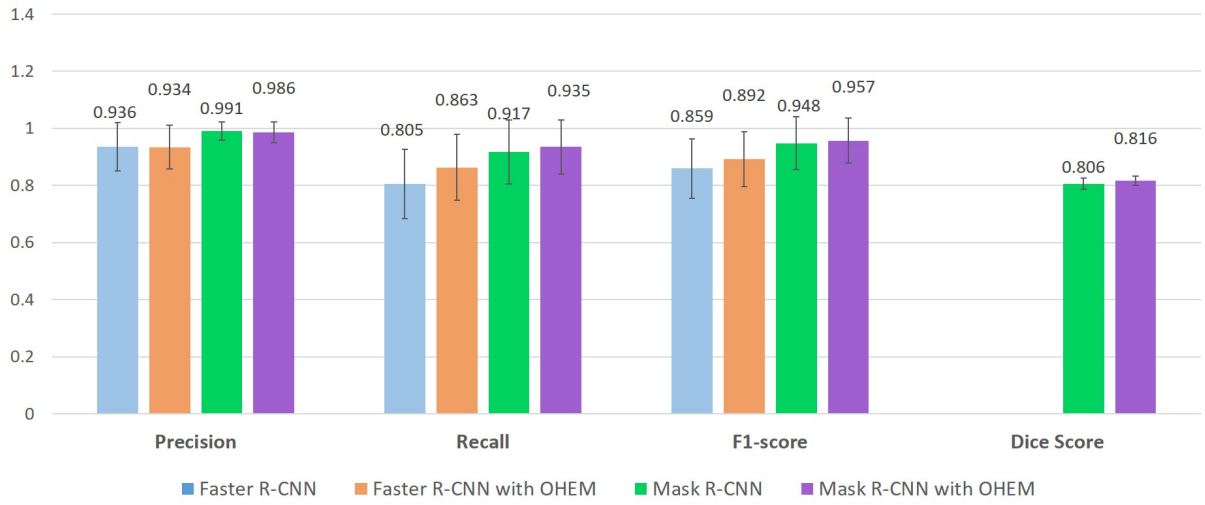

Figure 14. Experimental results on all approaches.

Figure 14 demonstrates a bar chart, showing the experimental results on different approaches. Note that since Faster R-CNN and Faster R-CNN with OHEM can only handle with object detection task, there is no Dice Score evaluation for these two methods. The exact value for each performance metric is shown on the top of each bar, along with an error bar showing the standard deviation.

In order to better evaluate the performance, the cross validation strategy is performed on all approaches. In k-fold cross validation, the original sample is randomly partitioned into k equal-sized subsamples. A single subsample is used as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. Then this procedure is repeated k times, where each subsample is used as validation data only once. Finally, the calculation is performed by averaging over k results. In particular, 3-fold cross validation is applied in the experiments.

Table 12. A comparison between different approaches on cross validation results.

Network	Precision	Recall	F1-score	Dice Score
Faster R-CNN	**0.963 (±0.056)**	0.815 (±0.123)	0.862 (±0.087)	--
Faster R-CNN with OHEM	0.952 (±0.064)	0.817 (±0.115)	0.875 (±0.081)	--
Mask R-CNN	0.952 (±0.099)	0.942 (±0.089)	0.940 (±0.074)	0.787 (±0.048)
Mask R-CNN with OHEM	0.950 (±0.095)	**0.947 (±0.083)**	**0.941 (±0.072)**	**0.789 (±0.081)**

Table 12 shows a comparison between different approaches using cross validation. Note that after augmenting Faster R-CNN with OHEM, the Recall value increases slightly by 2%. And the Recall value increases by 5% after augmenting Mask R-CNN with OHEM. Moreover, Mask R-CNN framework outperforms the Faster R-CNN framework, especially in Recall and F1-score, which approves that assumption that made in earlier chapter: the mask branch enables more information for object detection performance. Finally, this table also shows the mask prediction results given by Mask R-CNN and Mask R-CNN with OHEM. However, the dice score only increases by 2% after augmenting Mask R-CNN with OHEM.

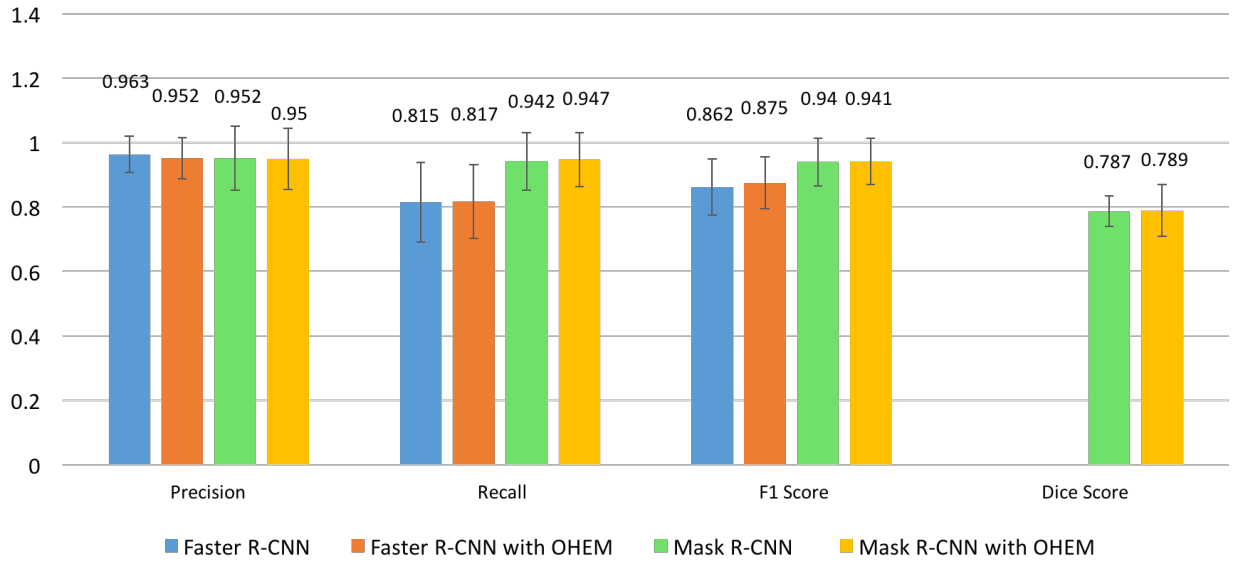

Figure 15. Cross validation results on all approaches.

Figure 15 shows the cross validation results on all approaches. Again, the exact value for each performance metric is shown on the top of each bar, along with an error bar showing the standard deviation.

Furthermore, a T-test is performed. A T-test is an analysis framework used to determine the difference between sample means. A T-test with two samples is commonly used with small sample sizes, testing the difference between the samples when the variances are unknown. In particular, a function ttest_ind in scipy library is used to calculate the T-test for the means of two independent samples. This is a test for the null hypothesis that two independent samples have identical average values. The T-test is performed to compare the difference between Faster R-CNN and Faster R-CNN with OHEM, the difference between Mask R-CNN and Mask R-CNN with OHEM, the difference between Faster R-CNN and Mask R-CNN, and the difference between Faster R-CNN with OHEM and Mask R-CNN with OHEM, respectively.

Table 13. T-test result between Faster R-CNN and Faster R-CNN with OHEM.

Evaluation Metric	Result on fold1		Result on fold2		Result on fold3		Average	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
Precision	-1.74	0.087	-0.743	0.46	0.151	0.88	-0.777	0.476
Recall	2.358	0.021	0.397	0.693	-0.415	0.68	0.78	0.465
F1-score	1.626	0.109	0.172	0.864	-0.258	0.797	0.513	0.59

Table 14. T-test result between Mask R-CNN and Mask R-CNN with OHEM.

Evaluation Metric	Result on fold1		Result on fold2		Result on fold3		Average	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
Precision	0.178	0.859	0.104	0.917	0.132	0.9	0.138	0.892
Recall	-0.26	0.793	-0.791	0.432	0.001	0.999	-0.177	0.741
F1-score	-0.04	0.966	-0.325	0.746	0.043	0.966	-0.107	0.893

Table 15. T-test result between Faster R-CNN and Mask R-CNN.

Evaluation Metric	Result on fold1		Result on fold2		Result on fold3		Average	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
Precision	1.014	0.314	1.313	0.196	-1.443	0.154	0.295	0.221
Recall	-7.043	0.000	-6.296	0.000	-3.815	0.000	-5.718	0.000
F1-score	-5.527	0.000	-4.365	0.000	-3.267	0.002	-4.386	0.001

Table 16. T-test result between Faster R-CNN with OHEM and Mask R-CNN with OHEM.

Evaluation Metric	Result on fold1		Result on fold2		Result on fold3		Average	
	t-value	p-value	t-value	p-value	t-value	p-value	t-value	p-value
Precision	0.180	0.859	0.998	0.322	-1.512	0.136	-0.111	0.439
Recall	-4.805	0.000	-7.192	0.000	-3.826	0.000	-5.274	0.000
F1-score	-3.059	0.004	-4.821	0.000	-3.440	0.001	-3.773	0.002

Table 13-16 show the T-test result between Faster R-CNN and Faster R-CNN with OHEM, and the T-test result between Mask R-CNN with Mask R-CNN with OHEM, the T-test result between Faster R-CNN and Mask R-CNN, and the T-test result between Faster R-CNN with OHEM and Mask R-CNN with OHEM, respectively.

The T-test measures whether the average value differs significantly across samples. If we observe a large p-value, for example larger than 0.05 or 0.1, then we cannot reject the null hypothesis of identical average scores. While if the p-value is smaller than the threshold, e.g. 1%, 5% or 10%, then we reject the null hypothesis of equal averages. On one hand, we observe that in both Table 13 and Table 14, the p-values are larger than 0.1 for each evaluation metric, suggesting that the difference is not significant across two approaches (i.e., Faster R-CNN versus Faster R-CNN with OHEM, and Mask R-CNN versus Mask R-CNN with OHEM). In other words, augmenting the detection and segmentation framework with OHEM does not significantly improve the performance of detection and segmentation. On the other hand, we can observe that in both Table 15 and Table 16, the p-values for Precision are larger than 10%, while the p-values for both Recall and F1-score are smaller than 1%. We can conclude that Mask R-

CNN does not significantly improve the Precision, compared to Faster R-CNN, while Mask R-CNN significantly outperforms Faster R-CNN in both Recall and F1-score.

4.4 Applications

There are two applications of fruit detection and segmentation. One is tomato counting for yield estimation, the other is size measurement.

4.4.1 Tomato counting for yield estimation

On one hand, we can detect how many tomatoes are there in one single image. Since we are able to obtain the accurate information of individual tomato location, it is possible to count the number of tomatoes, thus, we can further perform yield estimation and mapping. Ultimately, we may be able to design an automated harvesting system in the greenhouse at UGA in the future, given the hardware facilities.

$$|DiC| = |Count_{gt}\text{-}Count_{pd}| \qquad (7)$$

Table 17. The absolute Difference in Count for different approaches.

Network	$\|DiC\|$
Faster R-CNN	2.269 (±1.589)
Faster R-CNN with OHEM	1.769 (±1.394)
Mask R-CNN	1.154 (±1.347)
Mask R-CNN with OHEM	**0.731 (±1.041)**

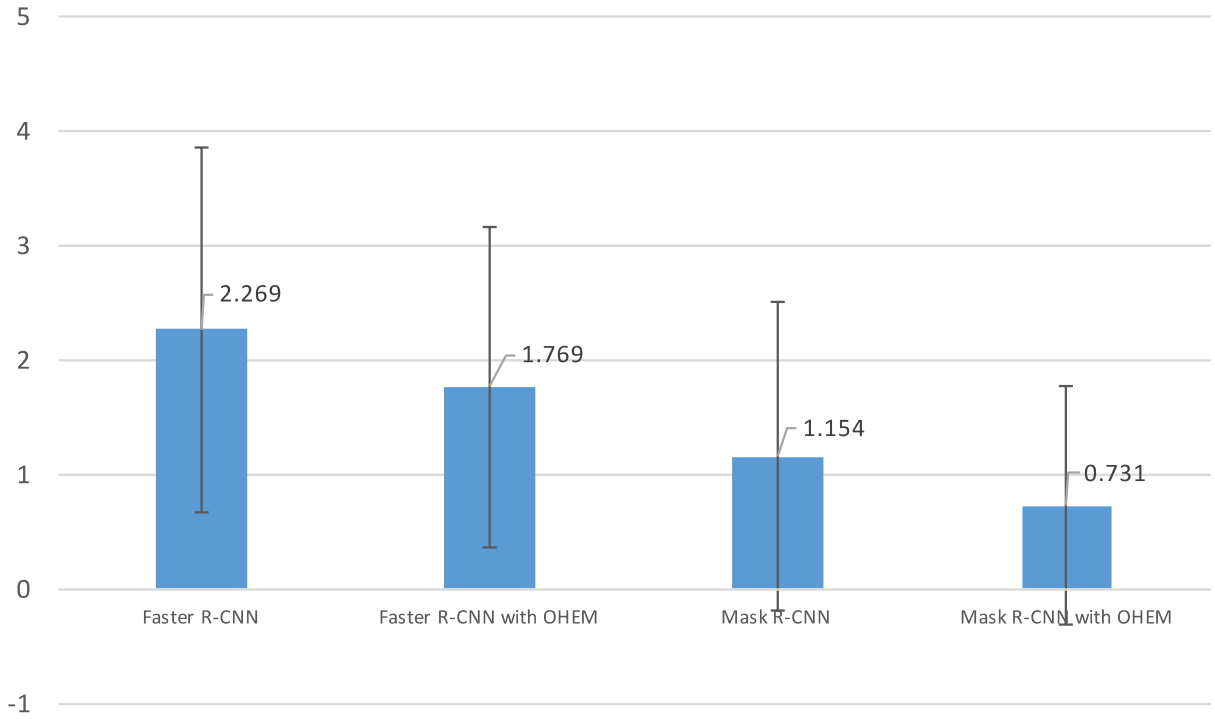

Figure 16. The absolute Difference in Count for different approaches.

Table 17 shows |DiC| for different approaches. $Count_{gt}$ is the true number of tomatoes in one image, and the $Count_{pd}$ is the predicted number of tomatoes in one image. Note that $Count_{pd}$ is also shown on the top-right corner of each image. |DiC| is calculated as in equation (7). The |DiC| is averaged over all test images. The standard deviation is also given along with |DiC|. It is clear to conclude that Mask R-CNN with OHEM achieves the lowest |DiC| among all approaches. Figure 16 shows a bar chart, representing the comparison on the absolute Difference in Count.

Similarly, Table 18 and Figure 17 demonstrate the comparison on the absolute Difference in Count after applying 3-fold cross validation.

Table 18. The absolute Difference in Count using cross validation.

Network	$\|DiC\|$
Faster R-CNN	2.688 (±1.719)
Faster R-CNN with OHEM	2.210 (±1.863)
Mask R-CNN	1.278 (±1.574)
Mask R-CNN with OHEM	**1.254 (±1.534)**

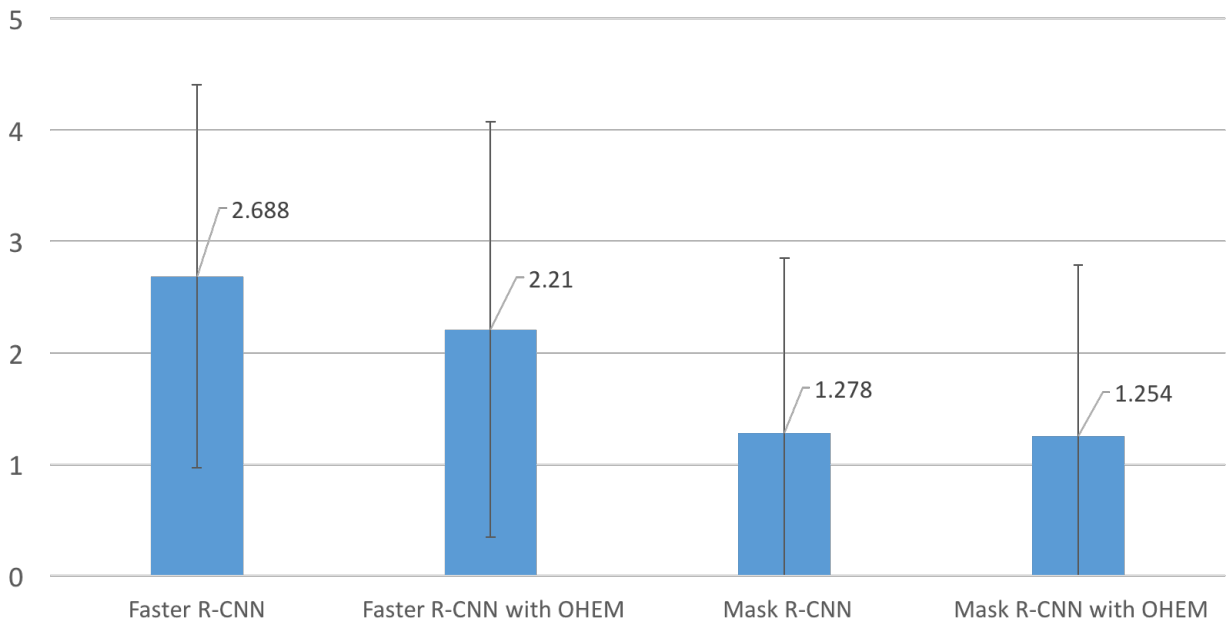

Figure 17. The absolute Difference in Count using cross validation.

4.4.2 Size measurement

On the other hand, since we have the predicted mask for each instance of tomatoes, it is possible to measure the growth of tomatoes over time by calculating the relative size for each tomato instance. Figure 18 shows the growth trajectories for each tomato instance over the tomato growth period (i.e., 32 days), starting from 12/12/2017 to 01/12/2018.

In Particular, the X axis demonstrates the day within the growth period, while the Y axis shows the relative size of the tomato instances (i.e., the number of pixels of the segmented mask). Note that the values on the right hand side of the plot illustrate the growth rate of each tomato instance. Also, note that the growth trajectories were drawn using robust linear regression.

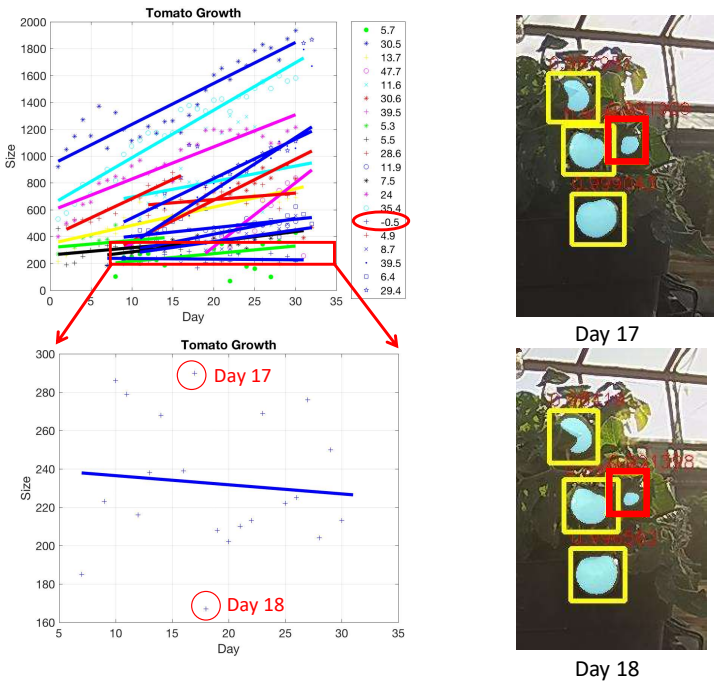

Figure 18. Growth trajectories for each tomato within 32 days. Note that there is only one growth trajectory with a negative growth rate. The details are shown in the plot on the bottom left.

Note that all trajectories illustrate that the relative size of each tomato instance keeps increasing at different growth rates, except one on the bottom right. The figure at the bottom left shows the details of the growth trajectory of this tomato instance. In particular, I highlighted two

points with red circle for day 17 and 18, respectively. Also the corresponding predicted results are shown on the right hand side of the figure.

After examining this tomato instance over time, I observer that this tomato instance is covered by leaves. In some specific dates, it's covered by leaves significantly, however, in other dates, it's slightly covered by the leaves. The reason for this is straightforward: as the tomato grows, the weight increases. So the tomato starts falling down towards the ground, which means the location of this tomato instance changes over time.

Therefore, we might want to propose an algorithm in the future to deal with the occlusion problem for this tomato datasets. Occlusion problem also exists in other fruit datasets. Once we are able to infer the occluded part of the fruit, it becomes more accurate to calculate the number of white pixels as the measurement of fruit size.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In summary, four different approaches are applied in this thesis. Specifically, Faster R-CNN and Faster R-CNN with OHEM are frameworks tackling with fruit detection task, achieving 0.859 and 0.892 F1-score on our tomato dataset respectively. Furthermore, Mask R-CNN and Mask R-CNN with OHEM manage to predict the fruit location and segmentation mask simultaneously in an end-to-end manner, achieving 0.948 and 0.957 F1-score respectively. Moreover, K-fold cross validation is performed on all approaches, resulting in a 2% increase in Recall by augmenting Faster R-CNN with OHEM, and a 5% increase in Recall by augmenting Mask R-CNN with OHEM.

To conclude, augmenting Faster R-CNN and Mask R-CNN with OHEM can improve the detection performance. Furthermore, the prediction of both tomato counting and size measurement is promising.

Future work could potentially include several directions. Firstly, as shown in section 4.4.2, most of predictions demonstrated by the growth trajectories are promising, except for one tomato instance because of the occlusion problem. Even though the occlusion problem is inevitable in the real world situation, it is still possible to measure the actual size of the fruits by inferring the occluded parts. Therefore, a potential future direction could be proposing an algorithm to infer the areas that are partially covered by leaves. Since we are measuring the growth of tomatoes by counting the number of pixels within the segmentation, it is difficult and inaccurate to measure the size of the tomatoes if they are mostly covered by leaves.

Secondly, since we have a set of images that are taken by the cameras every a half hour, it is intuitive to come up with the idea to take advantage of the time information. Specifically, the objects or fruits in the images taken in the same day tend to have higher similarity including color and shape information. Moreover, even for the images taken at the same time but different days, the lighting condition tends to be similar as well. For example, the images taken at 9 am are more likely to be darker than those taken at noon. Therefore, time information can potentially be utilized for future research. In my opinion, in the future work, a LSTM model can be added to the existing network architecture as an additional branch for time information.

Thirdly, weakly-supervised learning method for object detection and segmentation is also a potential direction. Since the ground truth annotation, especially for segmentation, is time-consuming and requires human labor, a weakly or semi-supervised learning algorithm is desired for this task in the future. In particular, a weakly-supervised framework can be proposed based on Generative Adversarial Network (GANs) [49]. A GAN contains a generator network to provide extra training examples to a classifier, which plays a role as a discriminator in the GAN framework. Since we can obtain the detection performance with F1-score of above 90%, we are able to localize the objects precisely. Inspired by the idea of GAN, we want to leverage the Mask R-CNN framework without providing the ground truth mask, but only the bounding box annotation. Thus, the generator can output fake images for the discriminator given the pixel-level class annotation from the mask branch.

REFERENCE

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *NIPS,* 2015.

[2] A. Payne, K. Walsh, P. Subedi, and D. Jarvis, "Estimating mango crop yield using image analysis using fruit at 'stone hardening' stage and night time imaging," *Computers and Electronics in Agriculture,* vol. 100, pp. 160-167, 2014.

[3] S. Bargoti, "Deep fruit detection in orchards," *ICRA,* 2017.

[4] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, "DeepFruits: A Fruit Detection System Using Deep Neural Networks," *Sensors,* vol. 16, 2016.

[5] Z. B. Husin, A. Y. B. M. Shakaff, A. H. B. A. Aziz, and R. B. S. M. Farook, "Feasibility Study on Plant Chili Disease Detection Using Image Processing Techniques," in *ISMS*, 2012, pp. 291-296.

[6] T. Brosnan and D.-W. Sun, "Improving quality inspection of food products by computer vision—a review," *Journal of Food Engineering,* vol. 61, pp. 3-16, 2004/01/01/ 2004.

[7] A. Shrivastava, A. Gupta, and R. Girshick, "Training Region-based Object Detectors with Online Hard Example Mining," *CVPR,* 2016.

[8] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *ICCV,* 2017.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CVPR,* 2014.

[10] K. E. A. v. d. S. J.R.R. Uijlings, T. Gevers, A.W.M. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision,* vol. 104, pp. 154-171, 2013.

[11] J. hosang, R. Benenson, P. Dollar, and B. Schiele, "What Makes for Effective Detection Proposals?," *TPAMI,* vol. 38, 2015.

[12] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS,* 2012.

[13] Y. LeCun, B. Boser, J. S. Denker, and D. Henderson, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* vol. 1, 1989.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *ECCV,* pp. 346-361, 2014.

[15] R. Girshick, "Fast R-CNN," *ICCV,* 2015.

[16] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *CVPR,* 2015.

[17] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi*, et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," *CVPR,* 2017.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CVPR,* 2016.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu*, et al.*, "SSD: Single Shot MultiBox Detector," *ECCV,* pp. 21-37, 2016.

[20] A. Garcia-Garcia, S. Orts, S. Oprea, V. Villena-Martinez, and J. G. Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation," *CoRR,* 2017.

[21] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *ICLR,* 2013.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *MICCAI,* pp. 234-241, 2015.

[23] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous Detection and Segmentation," *ECCV,* pp. 297-312, 2014.

[24] B. Hariharan, P. Arbelaez, and R. Girshick, "Hypercolumns for Object Segmentation and Fine-grained Localization," *CVPR,* 2015.

[25] J. Dai, K. He, and J. Sun, "Convolutional Feature Masking for Joint Object and Stuff Segmentation," *CVPR,* 2015.

[26] P. O. Pinheiro, R. Collobert, and P. Dollar, "Learning to Segment Object Candidates," *NIPS,* 2015.

[27] J. Dai, K. He, and J. Sun, "Instance-aware Semantic Segmentation via Multi-task Network Cascades," *CVPR,* 2016.

[28] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-aware Semantic Segmentation," *CVPR,* 2016.

[29] A. Kirillov, E. Levinkov, B. Andres, B. Savchynskyy, and C. Rother, "InstanceCut: from Edges to Instances with MultiCut," *CVPR,* 2016.

[30] M. Bai and R. Urtasun, "Deep Watershed Transform for Instance Segmentation," *CVPR,* 2017.

[31] S. Liu, J. Jia, S. Fidler, and R. Urtasun, "SGN: Sequential Grouping Networks for Instance Segmentation," *ICCV,* 2017.

[32] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan*, et al.*, "Microsoft COCO: Common Objects in Context," *ECCV,* pp. 740-755, 2014.

[33] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, and F. Moreno-Noguer, "Fracking Deep Convolutional Image Descriptors," *ICLR,* 2015.

[34] X. Wang and A. Gupta, "Unsupervised Learning of Visual Representations using Videos," *ICCV,* 2015.

[35] I. Loshchilov and F. Hutter, "Online Batch Selection for Faster Training of Neural Networks," *ICLR,* 2016.

[36] Y. Zhu, R. Urtasun, R. Salakhutdinov, and S. Fidler, "segDeepM: Exploiting Segmentation and Context in Deep Neural Networks for Object Detection," *CVPR,* 2015.

[37] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN Features off-the-shelf: an Astounding Baseline for Recognition," *CVPR,* 2014.

[38] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *NIPS,* 2014.

[39] K. Yamamoto, W. Guo, Y. Yoshioka, and S. Ninomiya, "On Plant Detection of Intact Tomato Fruits Using Image Analysis and Machine Learning Methods," *Sensors,* vol. 14, p. 12191, 2014.

[40] J. A. Hartigan and M. A. Wang, "A K-means clustering algorithm," *Applied Statistics,* vol. 28, 1979.

[41] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks For Large-Scale Image Recognition," *ICLR,* 2015.

[42] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick*, et al.*, "Caffe: Convolutional architecture for fast feature embedding," *Proceedings of the 22nd ACM international conference on Multimedia,* 2014.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CVPR,* 2016.

[44] T. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," *CVPR,* 2017.

[45] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," *NIPS,* 2015.

[46] M. M. H. Scharr, A.P. French, C. Klukas, D.M. Kramer, X. Liu, I. Luengo, J.M. Pape, G. Polder, D. Vukadinovic, X. Yin,S.A.Tsaftaris, "Leaf segmentation in plant phenotyping: a collation study," *Machine Vision and Applications,* vol. 27, pp. 585-606, 2016.

[47] D. P. Kingma and J. L. Ba, "Adam: A Method For Stochastic Optimization," *ICLR,* 2015.

[48] L. Bottou, "Stochastic Gradient Descent Tricks," in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421-436.

[49] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair*, et al.*, "Generative Adversarial Nets," *NIPS,* 2014.