A Strawberry Detection System Using Convolutional Neural Networks

Nikolas Lamb
Computer Science
Clarkson University
Potsdam NY, USA
lambne@clarkson.edu

Mooi Choo Chuah
Computer Science
Lehigh University
Bethlehem PA, USA
mcc7@lehigh.edu

***Abstract*—In recent years, robotic technologies, e.g. drones or autonomous cars have been applied to the agricultural sectors to improve the efficiency of typical agricultural operations. Some agricultural tasks that are ideal for robotic automation are yield estimation and robotic harvesting. For these applications, an accurate and reliable image-based detection system is critically important. In this work, we present a low-cost strawberry detection system based on convolutional neural networks. Ablation studies are presented to validate the choice of hyper-parameters, framework, and network structure. Additional modifications to both the training data and network structure that improve precision and execution speed, e.g., input compression, image tiling, color masking, and network compression, are discussed. Finally, we present a final network implementation on a Raspberry Pi 3B that demonstrates a detection speed of 1.63 frames per second and an average precision of 0.842.**

Keywords—precision agriculture, deep learning, fruit detection

I. Introduction

As agricultural automation becomes more feasible, due in part to lighter and more robust computer vision algorithms, produce detection has emerged as an essential part of the production pipeline. Automatic harvesting of fruit requires a precise understanding of individual fruit location, and delicate fruit may require systems to be aware of additional attributes like color or shape. Fundamental tasks like yield estimation and resource management also rely on having accurate knowledge of fruit location. These tasks are expensive and time-consuming when performed manually, and mishandling of produce by workers can often result in damaged produce [1]. Automation, through use of a produce detector, provides a very cost-effective method of processing produce and mitigates the risk of damage by improper handling.

There exist several approaches to detecting fruit, most of which utilize convolutional neural networks (CNNs), as color cameras are inexpensive and make capturing data over large swaths of field easy. However, existing implementations capture images which contain very dense patches of small fruits, a configuration which lends to large, complex networks. Existing implementations also favor high resolution images to maximize learned features, especially with very small fruits. Because of this, these systems require expensive hardware to operate, or are unable to predict quickly on most hardware.

This work presents a sparse CNN optimized to run quickly on mobile low-power hardware. Our approach is novel as it applies many optimization techniques to enable the system to run on computationally limited hardware. Our network implements Single Shot Multibox Detector (SSD) [2], a state-of-the-art detector framework which maximizes both speed and precision. Our system is designed to process low density images of strawberries, allowing us to experiment with exhaustive input reduction methods and techniques to exploit color and location to produce better predictions. These optimizations enable our system to predict quickly when deployed on a highly mobile low-power single board computer (SBC).

This paper is organized as follows: In Section II, we provide a brief overview of existing fruit detection systems and network optimization methods. In Section III, we provide an overview of our proposed CNN for detecting strawberries. In Section IV, we discuss the techniques implemented to optimize the CNN, which enables the network to run efficiently on low-cost off the

Funded by the National Science Foundation under grant no. CNS-1757787.

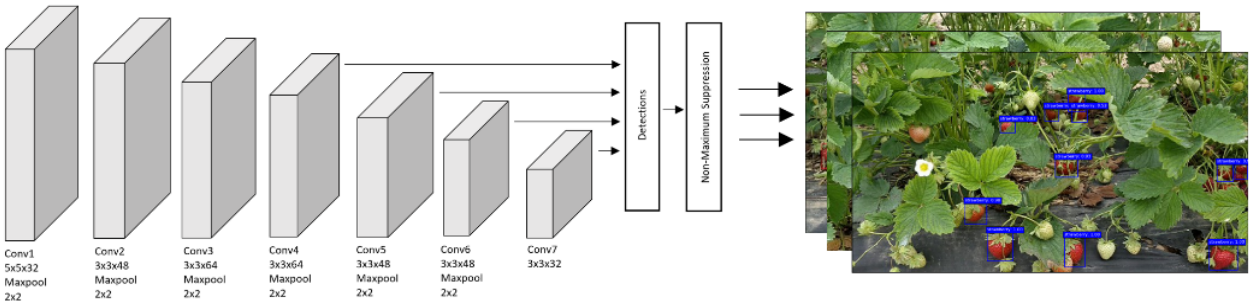

Fig. 1. Our initial SSD network configuration. The first 3 convolutional layers perform basic image classification. Successive layers generate feature maps, establish multi-scale predictions and generate classification confidences. The final predictions are then pruned using non-maximum suppression.

shelf devices such as the Raspberry PI 3B SBC. In Section V, we describe our experimental setup, the dataset we use, and present several ablation studies performed on our dataset. Finally, in Section VI we present our final CNN configuration and discuss how our designed fruit detector can be incorporated into a low-cost robot for fruit harvesting.

II. Related Work

Several methods have already been proposed and tested to identify fruit on a large scale for automating agricultural tasks. Most prior approaches utilize a state-of-the-art localization, segmentation, or classification network to identify one or more types of fruit in their growing habitat. Approaches generally favor high resolution images, allowing more fruit to be included in each image and maximizing learned features. Additional pre or post-processing techniques are sometimes employed, such as Region Of Interest (ROI) segmentation or color masking, to identify zones which have a higher probability of containing fruit [3]. Various compression techniques can also be applied to the input to reduce computational workload. Combinations of these processes have the potential to both accelerate prediction speed and increase network accuracy.

Bargoti et al. describe a segmentation network for fruit detection in various settings [4], [5] using feature learning algorithms, tested on apples and almonds. This approach gives precise predictions, identifying individual fruit on a pixel-wise basis. Various heuristic measures are then used to classify them as correct or incorrect. Linker et al. describes blob extraction [6], which allows individual mangos to be resolved when occluded or clustered by matching transformed circular zones to proposed mangos. Similarly, Payne et al. analyzes colors and textures to produce pixel-wise classifications on mangos, which are then refined using blob extraction [7]. Shape analysis can also be employed to verify proposed fruit regions [8], as shown by Yamamoto et al., who extracts circular apples from segmented predictions. Nuske et al. also uses radial symmetry of specular reflections to extract key points on berries [9].

Though many algorithms which focus on image segmentation exist, object detection using bounding boxes has become increasingly used for piecemeal identification, as bounding box annotations are much easier to generate. Several more optimized detectors which implement bounding box detection (R-CNN, Faster R-CNN, YOLO) have recently been shown to achieve high accuracy on the PASCAL-VOC detection dataset [10]. Further optimizations to Faster R-CNN using fully convolutional networks have shown additional speed and accuracy improvements [11]. Faster R-CNN has been used to detect densely clustered almonds, mangos, and apples [12] in high resolution images, and has also been used for sweet pepper and rockmelon detection when shown with five other unknown fruit types [13]. However, these implementations require expensive desktop computers to run in real time. Our approach is novel as it produces accurate predictions while running on computationally limited low-cost hardware.

III. Fruit Detector

Though Faster R-CNN has been most frequently used for fruit detection, SSD has been shown to give similar accuracy results and is able to generate predictions significantly faster [2], which is why it was implemented for our system. This section presents an overview of our SSD framework for fruit detection. It also includes brief descriptions of common techniques for improving speed and accuracy that were used in the base network implementation or were used in our network.

A. Network Architecture

Our network, shown in Fig. 1, is based on the framework proposed by Liu et at. [2]. As in the original implementation, our network is composed of four primary modules:

1) *Base network image classifier*
   A set of convolutional layers for basic image classification.
2) *Multi-scale feature maps for detection*
   Additional convolutional feature layers of progressively decreasing size which augment the truncated base network to allow predictions of detections at multiple scales.
3) *Convolutional predictors for detection*
   A set of convolutional feature layers which generate a fixed set of predictions using a set of convolutional filters.
4) *Default boxes and aspect ratios*
   Layers that generate and associate default bounding boxes with feature map cells and compute per-class scores for each identified bounding box.

During training, the input to the network is a set of three-channel BRG color images with associated rectangular ground truth bounding boxes. A set of prediction boxes over a variety of locations, aspect ratios, and scales is generated and compared to ground truth boxes using Jaccard overlap. Considering the substantial number of boxes generated by this method, it is essential to perform non-maximum suppression (NMS) efficiently during inference. This can be done by choosing an appropriate confidence threshold and a suitable Jaccard overlap threshold to filter out less relevant boxes.

B. Transfer Learning

Instantiating weights from a pretrained network has become an established method to train new, highly accurate networks. Other approaches have used this technique to create fruit detection networks which exhibit high accuracy [12]. The original implementation of SSD recommends using VGG-16 as a base network to perform general object classification. However, a robust network like VGG-16 is both large and somewhat slow to predict. Because our implementation utilizes low-power hardware, this must be taken into account when choosing a classifier. Therefore, to accelerate predictions our network does not take advantage of transfer learning, instead using weights which are randomly initialized. Our network is also significantly smaller than most state-of-the art classifiers, with only three dedicated convolutional classification layers.

C. Data Augmentation

Enlarging datasets by performing geometric or color

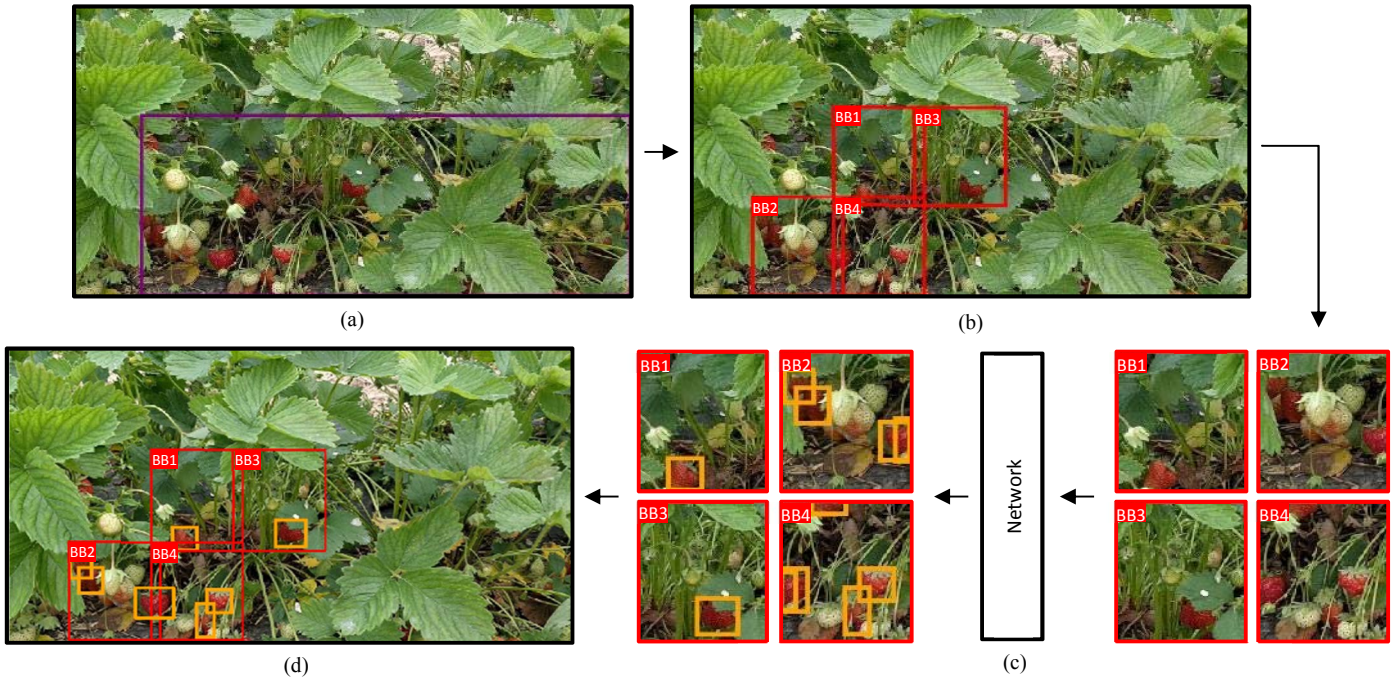

Fig. 2. Image tiling method of input reduction. (a) Purple bounding box shows ROI image generated using color mask. (b) Red bounding boxes show overlapping sub-images that contain valid color values. (c) Orange bounding boxes show strawberry predictions after sub-images are sent to the predictor. (d) Predictions are adjusted based on the location of their corresponding sub-image within the input frame.

transformations is an effective way to create more robust and adaptable networks. Our data generator uses a sequence of geometric transformations to augment the data during training. This allows the network to correctly identify fruits from a variety of angles and occlusions, as may be observed during field traversal. However, the data generator generally preserves the input color, as this is a more vital indicator of fruit condition. Fruits in particular exhibit specific colors at various stages of maturity, and for this reason generating filters that can accurately identify them requires preserving their initial color.

D. Hardware

A frequent concern of industrial consumers when considering adopting an emerging technology is its high initial cost. This cost is usually a result of prolonged development of software or hardware. As shown in the related works section, several methods already exist to identify fruits from densely populated images, but their high prediction speed can only be achieved on high-end computers. In this project, we desire to take advantage of existing low-cost hardware so that our deployment is inexpensive and efficient. To satisfy this, we optimize our fruit detector to run at a reasonable frame rate on a Raspberry Pi 3B low-power SBC, which is available from a variety of suppliers for $35. This off-the-shelf device is of interest since it has a 1.2GHz CPU and 1GB of RAM, supports USB peripherals and ethernet, has dedicated camera and display ports, and consumes a peak of only 2.5A at 5V when under load. Additionally, this computer can easily be attached to a field-traversing robot, taking up only about 100 cubic cm.

IV. CNN Optimization Techniques

As stated earlier, we design our fruit detector to run on a low-cost SBC like the Raspberry Pi 3B. In this section we explore the various optimization techniques required to achieve high inference speed while maintaining high precision.

A. Input Compression

Compressing the input to a neural network is one of the most effective methods of accelerating its inference speed. For convolutional neural networks, this can be achieved by down sampling the input images to a lower resolution. Reduction of the input dimensions by half results in an almost four times reduction in inference time, and this exponential trend continues with even higher compression factors. However, compressing the input also degrades the data, resulting in generally lower precision as the compression factor is increased. Even though our dataset is composed of images with relatively low object density, reducing the input dimensions too much resolves fruits to an unrecognizable degree, plummeting precision. It is therefore the goal of input compression to strike a balance between inference speed and relative precision. This objective is recurrant in other opimization methods as well.

B. Color Masking and Image Tiling

Like input compression, image tiling is another method of reducing the size of the network input. Image tiling relies on identifying regions of an image that have a higher chance of containing objects of interest. This allows regions of the image which are obviously background to be ignored by the predictor. As mentioned in the related works section, several methods exist to produce these regions of interest. Our system implements color masking, as ripe strawberries are a recognizable shade of red and background foliage is primarily green. Additionally, the implementation of SSD that our system utilizes can predict more quickly on batches of images than on singleton images when prediction times are averaged. Our

system exploits this property by segmenting the region of interest into many smaller sub-images which are then batched together and predicted on in parallel.

During operation, video frames are passed to the preprocessor sequentially as they are captured. The following steps, shown in Fig. 2, are performed to segment each frame:

1) *Mask Application*
   Applying a color mask, all red pixels in the image are identified, and coordinates of the horizontal and vertical extremities are extracted. These coordinates define the boundaries of an ROI image, Fig. 2 (a), which falls within the frame and includes every pixel identified by the mask.
2) *Image Segmentation*
   The ROI image is segmented into rectangular boxes which are approximately equal to a pre-determined resolution. Each box defines a sub-image within the ROI image, shown in Fig. 2 (b), that can be sent to the predictor. Each sub-image also overlaps neighboring sub-images by 10 pixels to ensure edge strawberries can still be identified. The mask is then applied to each sub-image, and sub-images which contain fewer than 100 red pixels are discarded. Remaining sub-images are sent to the predictor, Fig. 2 (c).
3) *Sub-Image Mask Application*
   Bounding box predictions are adjusted based on the locations of their corresponding sub-images within the original input frame, shown in Fig. 2 (d). To accommodate strawberries that occur between sub-images, bounding boxes on edges that overlap by 10 pixels are merged.

C. Network Compression

When deploying neural networks on SBCs, optimizing usage of computational resources is necessary to achieve reasonable performance. Network compression is used to reduce the number of parameters within the network, thus reducing computational load and increasing performance. Compression does not always increase inference speed however, so careful attention must be given to selecting the method of compression. Parameter pruning, low-rank factorization, compacted convolutional filters, and knowledge distillation [14] are some of the most common methods of network compression. Because of the relatively small size of our network, we implemented selective filter pruning over other compression methods.

To perform filter pruning, filters are extracted from each of the network's convolutional layers. Filters are then ranked in order of ascending L1-norm. Iterative L1-norm thresholding was used initially to remove filters, but we later found that removing batches of low-scoring filters instead produced consistently better results. Because of this, filters were later removed in groups of 16. To perform this method of compression, filters from each convolution layer are analyzed as before. However, rather than iterating over the filters in order of increasing L1-norm, if the 1st filter is found to have an L1-norm below the threshold, it and the next 15 lowest scoring filters are removed from the layer. This operation is repeated on the 17th filter, and so on, until the threshold is exceeded. Input dimensions for successive convolutional layers, class inference layers, and bounding box predictor layers are then resized according to the new dimensions of the filter batch. Finally, after filter removal each network is retrained using the same hyperparameters as were used to train initially.

V. Experimental Evaluations

Here, we present an overview of ablation studies performed on the object detection network, with detection results for the various network permutations. All trials use the same SSD network architecture, though in later trials we adjust and refine the structure of the network. Optimization techniques are applied sequentially to obtain the highest performance, with the more satisfactory networks from each successive method acting as test candidates for the next method. In the conclusion we present our final set of network optimization techniques

A. Experimental Setup

Our dataset is composed of high-resolution BRG images of ripe strawberries and manually annotated bounding boxes identifying each piece of fruit in each image. This image data was collected exclusively from a small strawberry farm near Bethlehem, PA. During data collection, a camera is held approximately two feet from a row of strawberry plants and set to collect video as the camera is moved down the row. Video is collected near peak daylight so that the strawberries are reasonably illuminated. Strawberry ripeness varies from unripe to overripe, though most of the strawberries captured are ripe enough to harvest. Strawberries are of a wide range of sizes and some exhibit unusual or uncharacteristic shapes. Strawberries are also shown occluded by other fruit, foliage, or dirt.

After video data is collected, the video is exported as a sequence of individual frames to be fed as input data to the network. To accelerate computation, the frames are compressed from their original resolution of 1080x1920 pixels (px) to 360x640px. Rectangular bounding boxes denoting ripe strawberries are manually generated. Because spatial continuity is maintained between video frames, bounding boxes can be slightly shifted from one to the next, simplifying the annotation process. Strawberries are annotated if they are (i) at least 25% visible, (ii) in the foreground, and (iii) appear to be ripe for picking. Relative ripeness is left to human interpretation, with the understanding that the network should avoid detecting strawberries which are not yet ripe. To normalize the process, all bounding boxes are generated by a single person.

For training, the dataset is shuffled and randomly divided into training, validation, and testing sets. We use a split of 60% training, 20% validation and 20% testing. From approximately three minutes of video, we extract 4,550 images containing 22,662 annotated strawberries, which results in a split of 2,730 training, 910 validation, and 910 testing images. Though some optimization methods will expand this dataset, the ratio of training, validation, and testing images is maintained.

To begin the training process, the images are fed into the base layer, which is composed of three convolutional layers with a successively increasing number of filters. Smooth L1 norm loss

False Positives *False Negatives* *True Positives*

Fig. 3. Example predictions made by our final network on the test set, with ground truth boxes in blue and prediction boxes in orange. Columns contain examples, from left to right, of several false positives, several false negatives, and several true positives.

and back propagation are applied end to end using Adam optimizer and final detection confidences are attuned by softmax loss. The network produces a fixed-size collection of bounding boxes and confidence scores for objects contained within the boxes. The final output returns 400 predictions, which are pruned using non-maximum suppression and a default probability threshold.

Training is performed on a desktop workstation with a Nvidia GTX 1080ti. Training is run for 200 epochs at 1000 steps per epoch, with a batch size of 32 images. Images are loaded and then saved as an uncompressed array in a contiguous block of memory to accelerate training times. The inference speed testing is performed on a Raspberry Pi 3B SBC. For testing purposes video rendering is disabled and all triggering is done remotely through secure shell (SHH).

B. Impact of Input Compression

To test input compression, images are compressed to various resolutions with linear color interpolation between pixels. Compression factors are chosen based on their proportional relationship to the base resolution. As original video is taken at a standard 16:9 aspect ratio, additional compression factors preserve this aspect ratio. This maintains geometric structure and spatial arrangement of fruits.

Table I shows detection results and prediction speeds for various input resolutions. It can be observed that precision decreases exponentially with resolution, while frame rate increases exponentially. When high resolution inputs are fed into the network, resulting detections are highly precise, nearing 0.877 average precision (AP) [10] at 360x640px. High resolution predictions also take significantly longer to generate, with an average frame rate of 0.23 frames-per-second (FPS) at the same resolution. Similarly, predictions on low resolution inputs show very low precision but very high prediction speeds, 0.518 AP and 3.16 FPS respectively at 72x128px.

TABLE I

Speed and Precision at Various Resolutions

Height (px)	Width (px)	Precision (AP)	Speed (FPS)
360	640	0.877	0.23
180	320	0.843	0.84
144	256	0.827	1.22
108	192	0.725	1.91
90	160	0.664	2.45
72	128	0.518	3.16

C. Impact of Color Masking and Image Tiling

Two different sub-image aspect ratios were tested, as altering aspect ratio of image tiles does not affect spatial relationship. To generate the color mask, we selected a range of HSV values to denote the color of ripe strawberries. Subsequent manual testing was performed to verify the color range. To train the network, our initial dataset is segmented according to the procedure described in Section IV B. To maintain the integrity of the ground truth boxes, bounding boxes are analyzed before and after segmentation and are discarded if segmentation has reduced their area by more than 80%. This may occur when bounding boxes fall between sub-images but have most of their area, and identifiable fruit features, in only one of the sub-images they are divided amongst.

Table II shows the detection results and prediction speeds for various sub-image resolutions. It can be observed that though precision only slightly decreases, from 0.880 to 0.814 AP, speed increases dramatically, from 0.48 FPS to 1.45 FPS, as the resolution is reduced from 144x256px to 72x72px. Although additional prediction speed gains could be achieved by continuing to reduce the resolution, strawberries would begin to fill the entire frame, severely handicapping feature learning. Because of this, precision begins to decrease more sharply at 72x72px, as the largest strawberry in the dataset has a bounding box of 58x79px.

TABLE II

Speed and Precision at Various Sub-Image Resolutions

Height (px)	Width (px)	Precision (AP)	Speed (FPS)
360	640	0.877*	0.23*
144	256	0.880	0.48
100	100	0.864	1.05
72	72	0.814	1.45

*This trial was generated using base input image without tiling as a control for tiled resolutions.

D. Impact of Network Compression

Table III shows the effect of network compression via filter removal on precision and prediction speed. As previously mentioned, because acceleration methods were tested sequentially, compression was applied to the best candidates from the image tiling method. It can be observed that removing more filters generally increases speed and decreases performance, though this is only evident when observing trends from a high level. Extreme compression rates see the removal of seventy percent of filters with very little precision degradation, from 0.864 to 0.842 AP in the case of 100x100px, indicating that many of the learned features are weighted very low. Finally, of the two tiling resolutions tested, 100x100px shows higher precision with the same number of filters removed, indicating that was able to identify more valuable features from the dataset than the smaller 72x72px network.

TABLE III

Speed and Precision for Various Compressions at 72x72 Resolution

Filters Removed	Network Parameters	Precision (AP)	Speed (FPS)
0	185,520	0.814	1.45
23	165,846	0.815	1.24
96	98,676	0.803	1.31
112	88,392	0.768	1.45
160	61,728	0.758	2.15
Speed and Precision for Various Compressions at 100x100 Resolution			
0	185,520	0.864	1.05
96	94,224	0.838	1.04
160	57,672	0.842	1.63

VI. Conclusion

Prior work has demonstrated the value of computer vision algorithms for fruit detection. Our system strives to be fast, precise, and affordable to spur easy adoption and high efficiency. We implement the state-of-the-art SSD neural network framework as a fruit detector. We use a sparse, three-layer convolutional classifier as our base classifier, and alter the network in various ways to increase speed and precision. To accelerate our network performance, we first compress the input to the network to 360x640px and apply a color mask to isolate regions of interest. The region is broken into many sub-images of 100x100px which are sent to the network in a single batch. Finally, the entire network is compressed by removing 160 lowly weighted filters, and then retrained.

Combining all optimization methods, we assembled a network with three convolutional layers (176 filters and 57,627 parameters) capable of predicting with 0.842 AP at 1.63 FPS on a Raspberry Pi 3B SBC. Example correct and incorrect predictions generated by this network are shown in Fig. 3. This network was trained on a version of our original dataset which was processed using the color masking and image tiling method discussed in Section IV B. This fruit detector will be used to detect strawberries for mass harvesting. As the model is given a live video, individual frames will be augmented with depth information to approximate the location of each identifiable strawberry in 3D. Predicted bounding boxes will then be used to generate a path for an articulated robot arm or picking apparatus. This system will enable unmanned robots to replace humans as harvesters, dramatically improving agricultural efficiency and reducing damage to crops.

Acknowledgements

We would like to thank Xiaowen Ying, a graduate student who offered valuable help with implementation techniques. This work was funded by the Lehigh NSF REU Site project: Intelligent & Scalable System under contract number 1757787.

References

[1] T. Gemtos, S. Fountas, A. Tagarakis, V. Liakos, "Precision agriculture application in fruit crops: Experience in handpicked fruits," *Procedia Technology,* vol. 8, pp. 324-332, 2013.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. C. Berg, "SSD: Single Shot MultiBox Detector," *CoRR,* 2015.

[3] S. Singh, M. Bergerman, J. Cannons, B. Grocholsky, B. Hamner, G. Holguin, L. Hull, V. Jones, G. Kantor, H. Koselka et al., "Comprehensive automation for specialty crops: Year 1 results and lessons learned," *Intelligent Service Robotics,* vol. 3, no. 4, pp. 245-262, 2010.

[4] S. Bargoti, J. P. Underwood, "Image Segmentation for Fruit Detection and Yield," *Journal of Field Robotics,* vol. 34, no. 6, pp. 1039-1060, 2016.

[5] C. Hung, J. Nieto, Z. Taylor, J. Underwood, S. Sukkarieh, "Orchard Fruit Segmentation using Multi-spectral Feature Learning," in *International Conference on Intellignet Robots and Systems*, Tokya, Japan, 2013.

[6] R. Linker, O. Cohen, A. Naor, "Determination of the number of green apples in RGB images recorded in orchards," *Computers and Electronics in Agriculture,* vol. 81, pp. 45-57, 2012.

[7] A. Payne, K. Walsh, P. Subedi, 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. 3, no. 1-2, pp. 4-34, 2012.

[8] K. Yamamoto, W. Guo, Y. Yoshioka, S. Ninomiya, "On plant detection of intact tomato fruits using image," *Sensors,* vol. 14, no. 7, pp. 12191-12206, 2014.

[9] S. Nuske, K. Wilshusen, S. Achar, L. Yoder, S. Singh, "Automated visual yield estimation in vineyards," *Journal of Field Robotics,* vol. 31, no. 5, pp. 837-860, 2014.

[10] M. Everingham, L. Van Gool, C. Williams, J. Winn, A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision,* vol. 88, no. 2, pp. 303-338, 2010.

[11] J. Dai, Y. Li, K. He, J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," *CoRR,* 2016.

[12] S. Bargoti, J. P. Underwood, "Deep Fruit Detection in Orchards," in *IEEE ICRA,* Singapore, 2017.

[13] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, C. McCool, "DeepFruits: A Fruit Detection System Using Deep Neural Networks," *Sensors,* vol. 16, no. 8, p. 1222, 2016.

[14] Y. Cheng, D. Wang, P. Zhou, T. Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks," *CoRR,* 2017.