Abstract. Automation of agricultural management has become important by an aging population and decreasing agricultural population. A deep neural network can be applied to detect vegetables and fruits on a farm. Detecting objects in an image taken from a camera leads to counting the objects. However, the training cost for the deep neural network is generally high, and it requires a high-quality large dataset and time consumption. Also, there are various kinds of vegetables, and it is not feasible to train and tune neural networks for all of them because creating datasets for all vegetables requires a high cost. Then, we propose the method which employs a few-shot model and data augmentation for few-shot and few-annotated datasets. Finally, we create the novel dataset for detecting eggplants as a dataset for our experiment and evaluation. As a result, the proposed method with the few-annotation and few-shot settings obtained 5 to 20 points improvement of accuracy than that of the Faster R-CNN with the few-shot settings on AP50, and the model converges earlier 50 - 200 epoch than that of Faster R-CNN. However, the few-shot model with the few-shot setting obtains a higher performance than the few-shot and few-annotation settings on AP50 and converging time.

Keywords: Object Detection, Few-shot Learning, Data Augmentation.

1. Introduction

In Japan, an aging population of agriculture is a severe problem and, artificial intelligence techniques are expected to solve the problem as an automated or autonomous agricultural system. Recent neural network studies show that convolutional neural network (CNN) based models are successfully used for object detection tasks. However, the CNN-based methods need large datasets with high-quality images and labels to train the model. In object detection, datasets creation squanders labor and time consumption because datasets have many annotations that are rectangle areas and class labels for the target object in each image. Large public image datasets for classification tasks and object detection tasks have already existed for the benchmark of neural models to train for general objects, PASCAL VOC [2] and COCO [11]. However, the datasets do not include specific field objects rare in real life, and fruiting crops in a farm are also relevant so that those objects are needed to create new datasets for each task.

Fine-tuning and few-shot learning is used in order to tackle this problem. The former requires only smaller datasets, and the latter also requires only a few images and labels. Then those methods need only a little time-consuming to train, like adjusting parameters for novel datasets because the models assume that those are pre-trained. However, those methods require at least one image with complete annotation as a training dataset.

An agricultural dataset for the practical situation is different from large public datasets, and the agricultural dataset has occlusions with overlapped leaves, the difference of crop scales, and complex background. Therefore, those contexts affect annotating objects by a human, and it is necessary to avoid annotation work.

In this paper, we propose the method that composes a few-shot model and data augmentation for a few-annotation and few-shot setting practical agricultural dataset and experiment. In the experiment, we correct a set of images, and we annotated manually for detecting eggplants task with a few-shot setting and a few-annotation setting. Then, we train a few-shot model by the augmented dataset that is entirely new compared to created original eggplant dataset. Finally, we evaluate detecting performance with the few-shot setting and the few-annotation setting by comparing exists object detection model with the few-shot setting and the few-shot model with the few-shot setting on object detection metrics and qualitative evaluation for the image with predictions.

In this research, we collaborate with the AI and IoT-powered agriculture research project, “IoP: the Internet of Plants”, and we use eggplant images for an agricultural CNN application in the project.

2. Related Works

2.1. Object Detection

CNN-based object detection architectures are roughly categorized into the following two types, one-stage model and two-stage model. The one-stage models split input images into a grid, and each cell predicts locations, confidences, and classes directly from features. Those models
are YOLO [15] and SSD [12] as typical examples. The two-stage models consist of an objectness region predictor and a final results predictor that uses the predicted region. In the first stage, those models generate anchors that indicate where it is possible to predict locations and predict location and objectness score, and predictions are to adjust each anchor. Those models also predict location and class from a region of interest (ROI) features in the second stage by adjusting first stage predictions. The two-stage models are R-CNN [6], Fast R-CNN [5], and Faster R-CNN [16] as examples. For example, Faster R-CNN has two trainable modules, a region proposal network (RPN) and a Fast R-CNN detector.

Generally, the two-stage models obtain higher prediction performance can be obtained stably than the one-stage ones, and one-stage ones are faster inference than the two-stage ones, but one-stage ones have recently improved prediction performance.

### 2.2. Few-shot Object Detection

The few-shot setting is a model train from few training data per class. This setting aims to adapt a model to predict novel objects with a small training dataset. In object detection, a model trains from a few images and annotations per category. In a few-shot setting, methods input exemplars per class called a support set and the query image into few-shot models that could predict unseen or rare objects, and the support set is a set of exemplars for all classes and query image represent a target object that is prepared per all target objects. Kans et al. and Pers et al. propose methods [10] and [14] that reweights intermediate features to adjust prediction by support images set, and that the methods need to generate weights for all classes from support set and could predict for all classes. Mics et al., Fans et al., and Hsis et al. propose methods [13], [3], and [8] that uses a query image to interact query image and target image, including target objects by an attention mechanism that direct attention between two features of the query and the target, and the methods require simply inputs and outputs for only one class. However, the methods predict for only one class per a forward path.

### 2.3. Data Augmentation

In training neural network models, amounts of data affect a generalization performance of a model but creating many data squander time consumption. Data augmentation is creating new data artificially from existing data, and models train with existing data and augmented data to aim for regularization. In classification tasks, MixUp [19] synthesizes two images, and CutOut [1] and Random erasing [20] hide part of an image, and CutMix [18] is combined MixUp and CutOut. In object detection, copy-paste augmentation [4] and patch-level augmentation [7] copy object image from an image and paste another image because a dataset for object detection requires not only class labels but also location labels.

### 3. Dataset

We create the agricultural dataset that is a collection of eggplant images and aims to detect fruited eggplants in the images. The images are taken in the greenhouse of Kochi Prefectural Agricultural Research Center in Japan. We take images with at least one eggplant without considering time, sunlight conditions, and appearance, and the image size (height, width) is (2592, 3456).

This dataset has 98 images. Train dataset extracts 5 images randomly from all images for five-shot settings and test dataset use remainings, 93 images.

The difficulty of detecting eggplants depends on because those images have differences in the scale of eggplant and occlusion by leaves. Then, we split the annotations by the area size of eggplant to create some datasets with different difficulties. Table 1 summarize that how object each dataset have. Figure 1 shows the example of objects for each difficulty.

Additionally, we assume the few-annotation setting that a dataset has only a few annotations for training a model, and we create the dataset for this setting so that some annotations are incomplete. In this setting, an object that has no annotation is treated as background. Considering annotating by humans, we decide that included annotations are only large-scale objects with no occlusion to avoid annotating for small and overlapped objects that are difficult to annotate in the few-annotation setting. By this operation, a dataset with the few-annotation and few-annotation setting has 5 annotations that fulfill the conditions of all 84 annotations for training.

### Table 1. The summary of the datasets. The column of min object size indicates the smallest area of an object included in each dataset.

<table>
<thead>
<tr>
<th>difficulty</th>
<th>min object size</th>
<th># of annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>easy</td>
<td>192 × 192</td>
<td>323</td>
</tr>
<tr>
<td>normal</td>
<td>96 × 96</td>
<td>685</td>
</tr>
<tr>
<td>hard</td>
<td>0 × 0</td>
<td>1155</td>
</tr>
</tbody>
</table>

### Fig. 1. The examples of objects for each difficulty. The easy dataset has red boxes, and the normal dataset has red and green ones, while the hard dataset has all boxes.
Algorithm 1 Generate negative patch bounding boxes

Input: Train images $I_i$ sizes (height and width) of $I_i$-th train image are $H_i$ and $W_i$. Number of negative patch $N_{neg}$, Crop image sizes $H_c$ and $W_c$

Output: Negative patch bounding boxes $B_{neg}$

1: for $i = 0 \ldots N_{neg}$ do
2: Get $h_c$ and $w_c$ from $H_c$ and $W_c$ randomly
3: Get $j$ from 0 to $i
$ randomly
4: Get $y_c$ and $x_c$ from (0, 0) to $(H_j - h_c, W_j - w_c)$ randomly
5: Append $[y_c, x_c, h_c, w_c]$ to $B_{neg}$
6: end for

Algorithm 2 Generate a patchwork image

Input: Positive patches $Patches_{pos}$, Negative patches $Patches_{neg}$, Grid size (height and width) $h_g$ and $w_g$, Patchwork image size $h_p$ and $w_p$, Probability $p$

Output: Patchwork image $I_{patchwork}$, Patchwork bounding boxes $B_{patchwork}$

1: $h_p, w_p = h_p / h_g, w_p / w_g$
2: for $i = 0 \ldots h_g$ do
3: for $j = 0 \ldots w_g$ do
4: Set $b_{patchwork}$ to $\{i \times h_p, j \times w_p, h_p, w_p\}$
5: if $p > Rand(0, 1)$ then
6: Get patch from $Patches_{pos}$ randomly
7: else
8: Get patch from $Patches_{neg}$ randomly
9: end if
10: Resize patch to $(h_p, w_p)$
11: Set patch to located $b_{patchwork}$ in $I_{patchwork}$
12: Append $b_{patchwork}$ to $B_{patchwork}$
13: end for
14: end for

Fig. 2. The patchwork image creation.

4. Method

We propose the method for the few-annotation setting and the few-shot setting. We focus on data augmentation.

4.1. Data Augmentation

The negative area occupies almost the dataset because it has only a few annotations. Moreover, when a model trains the dataset with objects with no annotations, it obstructs correct predictions. For this problem, we create new data from positives and negatives as data augmentation.

Dataset creation needs positive image patches and negative image patches. Positive image patches are partial images that have positive annotation. Negative image patches are partial images that are cropped with specific sizes randomly from images. Algorithm 1 shows how to create bounding boxes for generating negative patches. Negative image patches may include some positive image patches, so that we use k-means to exclude positives from negatives. We exclude negatives that k-means classified into the positive clusters. Finally, to create patchwork images, we concatenated patches selected randomly from positive and negative image patches as a grid and create annotations for positive image patches. Algorithm 2 shows how to generate a patchwork image. The new dataset has those patchwork images and annotations. Figure 2 shows how to create the patchwork image as an example.

4.2. Model Architecture

Because the dataset has only a few data for training a model, a model needs to be capable of training with few-shot settings. The agricultural dataset has only one category. Therefore, we use the architecture [8] that inputs target image and query image and output location for only one class in this situation. Figure 3 shows the structure of the model. This model is based on Faster R-CNN, and to the inputs target image and query image, this model has a siamese architecture that shares specific parameters.

The target image is including the target object, which is equivalent to input for default Faster R-CNN. The query image represents an example of the target object. The target image and query image are denoted by $I$ and $p$, respectively.

In the first stage, the siamese architecture backbone extracts features from $I$ and $p$. The features of $I$ are represented by $\phi(I) \in \mathbb{R}^{N \times W_I \times H_I}$, and the features of $p$ are represented by $\phi(p) \in \mathbb{R}^{N \times W_p \times H_p}$. When $\phi(I)$ refer to $\phi(p)$, results of non-local operation [17] is represented by $\psi(I; p) \in \mathbb{R}^{N \times W_p \times H_p}$. When $\phi(p)$ refer to $\phi(I)$, result of non-local operation is represented by $\psi(p; I) \in \mathbb{R}^{N \times W_I \times H_I}$. Then, features of $I$ and $p$ that is applied co-attention are presented by

\[
F(I) = \phi(I) \oplus \psi(p; I) \in \mathbb{R}^{N \times W_I \times H_I}, \quad \ldots \ldots \; (1)
\]

\[
F(p) = \phi(p) \oplus \psi(I; p) \in \mathbb{R}^{N \times W_p \times H_p}, \quad \ldots \ldots \; (2)
\]

and $\oplus$ operator is the element-wise sum for $\phi(\cdot)$ that is backbone feature and $\psi(\cdot)$ that is non-local features.

Next, this model use squeeze-and-excitation (SCE) [9] as an attention mechanism to obtain weighted features that are related to $F(I)$ and $F(p)$. In the “squeeze” step,
the MLP layer $\mathcal{M}$ transforms query image feature $p$, and global average pool (GAP) and global max pooling (GMP) summarize information for each channel of query feature to weight vector $w$ by summation of each result. The squeeze operation is represented by

$$w = \text{GAP}(\mathcal{M}(F(p))) + \text{GMP}(\mathcal{M}(F(p))) \in \mathbb{R}^N. \quad (3)$$

The co-excitation operation that is weighting $F(l)$ and $F(p)$ by $w$ is represented by

$$\tilde{F}(l) = w \odot F(l), \ldots \ldots \ldots \ldots \ldots \quad (4)$$

$$\tilde{F}(p) = w \odot F(p). \ldots \ldots \ldots \ldots \ldots \quad (5)$$

The $\odot$ operator is the channel-wise multiplication for each feature, $F(l)$ and $F(p)$ by $w$.

In the second stage, RPN generates $K$ region proposals from target non-local feature $F(l)$, and the number of region proposals is denoted as $K$. The ROIAlign extracts fixed-size features $R$ that relate to each region proposal from feature $\tilde{F}(l)$. The size of the extracted fixed-size feature is denoted as $S$. In the same way, ROIAlign transform overall $F(p)$ to a fixed-size feature $q$, that is the same architecture [3] at extracting the features in object level that is to take consistencies of extracting fixed-size feature way for both of target and query images. Those operations for $\tilde{F}(l)$ and $\tilde{F}(p)$ are represented by

$$R = \text{ROIAlign}(\tilde{F}(l), \text{RPN}(F(l)))) \in \mathbb{R}^{N \times K \times S \times S}. \quad (6)$$

MLP layer transform feature $R$ that is flattened each proposal features and feature $q$. Then, predictor operations concatenate each proposal feature $r \in \mathbb{R}^N, r \in R$ and feature $q$ to feature $x = [r^T; q^T]^T \in \mathbb{R}^{2N}$. All operated features are denoted as $X \in \mathbb{R}^{2N \times k}$. Finally, the predictor transforms $X$ to obtain locations and class scores.

In RPN, we use Smooth L1 loss and binary cross-entropy loss to optimize the model, that losses are denoted as $L_{\text{RPN}}$, and we also use smooth L1 loss $L_{\text{reg}}$ and binary cross-entropy $L_{\text{cls}}$ and margin-based ranking loss [8] $L_{\text{MR}}$ in the predictor. Finally, all losses are represented by

$$R = L_{\text{RPN}} + L_{\text{reg}} + \lambda L_{\text{MR}}, \ldots \ldots \ldots \ldots \ldots \quad (7)$$

and $\lambda$ is a coefficient.

5. Experiment

In the pre-training phase, we use train2017 in MS-COCO. Candidates for query images are only the objects that pre-trained Mask R-CNN available to predict. To make a pair of the target and query images, we randomly select a category included in an image and select a query image that is the same category from candidates randomly. We train with the same pairs in all epochs.

In training for the agricultural dataset, we use newly augmented datasets that 5 images are created from 5 shot settings agricultural dataset, and images size is $(512, 512)$, which grid size is $(4, 4)$, $(4, 4)$, $(8, 8)$, $(8, 8)$, $(16, 16)$. We generate positive patches and negative patches to create the dataset. To creating positive patches, we choose 5 positive ones that have no occlusion arbitrarily. To creating negative patches, we extract images that size is $(128, 128)$ randomly. To exclude positive patches from negative patches, we classify all patches into 8 clusters by k-means, and patches not in the positive cluster are negative patches. Patchwork image is created from resized patches that are picked from positive patches and negative patches randomly with probabilities for picking a positive patch that is $\{0.5, 0.5, 0.3, 0.3, 0.2\}$ for each patchwork image, and the resizing function is a bilinear interpolation.

It seems that the models with the few-annotation setting become underfitting or overfitting without data augmentation that removes no annotated eggplants. For this reason, we compare the few-shot model with the few-shot and few-annotation setting, the few-shot model with the few-shot setting, and Faster R-CNN with the few-shot setting. Table 2 shows a summary of each case in this experiment. The metrics that are to compare are AP50 and AR100. In the training of agricultural datasets for the few-shot model, the pairs of target and query are random, so that we perform the experiment for ten steps by changing the pairs for each trial and compute averages of AP50 and AR100 per epochs to be as results. We perform the
experiment with all difficulties.

The models have ResNet50 with a feature pyramid network as the backbone. In pre-training, the few-shot model train using Adam optimizer with momentum 0.9, weight decay 0.001, and learning rate $10^{-4}$ for 50 epochs, with batch size 32 on 8 GPUs in parallel, and we use $\lambda = 3$ for the margin-based ranking loss. Input image size is (512, 512). In fine-tuning, the few-shot model and Faster R-CNN train with a learning rate of $10^{-6}$ for 500 epochs, and other settings are the same as pre-training. We implement the models based on torchvision so that other hyper-parameters that are not described are the same as default Faster R-CNN in torchvision.

6. Discussion

Figure 4 show that the results of AP50 and AR100 for 500 epochs in each difficulty datasets.

For considering Faster R-CNN overfitting, we compare on 200 epoch. In the easy dataset on 200 epoch, AP of the few-annotation setting few-shot model is about 7 point higher than one of the few-shot setting Faster R-CNN, and AR100 of the few-annotation setting few-shot model is also about 2 point higher than one of the few-shot setting Faster R-CNN. Therefore, it seems that the few-annotation setting few-shot model predicts eggplant accurately more than Faster R-CNN.

In each difficulty, the few-shot model that trains the 5-shot setting dataset is the most performance, and the few-shot model that trains the few-annotation setting dataset to obtain more performance than the Faster R-CNN trains the 5-shot setting dataset on AP50. In each model, AP50 of the easy dataset is the highest performance, and one of the hard datasets is the lowest performance. The few-shot model is earlier to reach peak performance than Faster R-CNN in the 5-shot setting dataset.

Next, we evaluate qualitatively for the image with predictions. Figure 5 shows drawn predictions of the few-shot model with few-shot and few-annotation settings on 500 and 0 epoch. The model can predict large eggplants with a few occlusions by leaves, but the model cannot predict small or occluded others. Sometimes, the model predicts two eggplants that ground truth is correctly one eggplant and incomplete an eggplant that is only part of eggplant because of overlapping leaves. The cause of failing prediction for small and occluded is choosing available annotation for training that is selecting large eggplant with no occlusion so that the model can only predict similar eggplants with annotated clearly. In quantitatively and qualitatively evaluation, the few-shot models that are on 0 epoch without training achieve higher about 6 - 20 points than Faster R-CNN that is without training, and Figure 5(b) shows that the few-shot model with the few-annotation setting can predict some eggplants so that the few-shot models can without training.

7. Conclusion

In this study, we approach adapting CNN-based object detection to the practical agricultural situation for reducing consumption of creating dataset and training model from an aspect of the few-shot model and data augmentation. In the experiment, we create the dataset for detecting eggplants and train the few-shot model with the few-shot setting and the few-annotation setting. In the results, the few-shot model with the few-annotation setting obtains a higher AP50 than Faster R-CNN with the few-shot setting, and the AP50 is lower than the few-shot model with the few-shot setting, and the few-shot models converge faster than Faster R-CNN or without training.

Acknowledgements

We acknowledge Kochi Prefectural Agricultural Research Center in Japan for helping with the collection of agricultural data.

References:

The results of AP50 and AR100 are shown in Figures 4(a) and 4(b), respectively. Figure 4(a) displays AP50s per difficulties, while Figure 4(b) illustrates AR100s per difficulties. Blue lines represent the Faster R-CNN with the few-shot setting, orange lines indicate the few-shot model with the few-shot setting, and green lines depict the few-shot model with the few-shot and few-annotation setting.

Figure 5 demonstrates the drawn predictions and ground truths. Yellow boxes signify predictions, with the easy dataset featuring red boxes, the normal dataset having red and green boxes, and the hard dataset exhibiting all boxes. In other words, red, green, and blue boxes indicate large, medium, and small eggplants. Figure 5(a) presents predictions of the few-shot model with the few-annotation setting at 500 epochs, whereas Figure 5(b) shows predictions at 0 epochs.

References: