A Benchmark for Multi-Class Object Counting and Size Estimation Using Deep Convolutional Neural Networks

Zixu Liu[a,c], Qian Wang[b]*, Fanlin Meng[d]

[a]*Southampton Business School, University of Southampton, U.K.*
[b]*Department of Computer Science, Durham University, UK.*
[c]*Crop Science Centre, National Institute of Agricultural Botany, U.K.*
[d]*Alliance Manchester Business School, The University of Manchester, UK*
zixu.liu@soton.ac.uk, qian.wang173@hotmail.com, fanlin.meng@manchester.ac.uk

Abstract

Automatic object counting and object size estimation in digital images can be very useful in many real-world applications such as surveillance, smart farming, intelligent traffic systems, etc. However, most existing research mainly focus on scenarios where only one type of object is considered due to the lack of proper datasets. Furthermore, they use the traditional detection algorithms for size estimation and can only do segmenting tasks but cannot identify different types of objects and return corresponding individual size information. To fill these gaps, we create a synthetic dataset and propose a benchmark for multi-class object counting and size estimation (*MOCSE*) within a unified framework. We create the dataset *MOCSE13* by using Unity to generate synthetic images for 13 different objects (fruits and vegetables). Besides, we propose a deep architecture approach for multi-class object counting and object size estimation. Our proposed models with different backbones are evaluated on the synthetic dataset. The experimental results provide a benchmark for multi-class object counting and size estimation and the synthetic dataset can be served as a proper testbed for future studies.

Key words: Multi-class object counting, Crowd counting, Object size estimation, Convolutional neural networks, Synthetic dataset

1. Introduction

Object counting is a very common and challenging problem performed in different researches and industries where the task is to obtain the number of different types of objects in still images or videos. Counting manually is inefficient and time-consuming when the number of objects is large. Automated object counting which uses computer vision to replace manual methods has received significant attention in the last few years [10, 30]. After several years of development, automatic object counting in still images or videos (i.e. image sequences) becomes an important technique in many real-world applications. For example, one can use a drone to take photos of crowds from above and automatically get the number of people and vehicles in a place for traffic planning and risk control. In the applications of smart agriculture, one can estimate the count of plants, fruit or vegetable by simply taking photos or videos and feeding them into a model for automatic object counting. In many applications, not only does the count of objects matter but the sizes of objects in the images are also required (e.g., for yield estimation or object classification by sizes). As a result, there is an urgent requirement for developing approaches to automatic object counting and size estimation based on images.

1.1. Challenges and Research Questions

Object counting, especially crowd counting, has been studied extensively in recent years [34, 4, 2, 5, 39, 42]. The majority of works in this area have employed deep learning techniques and achieved impressive progress in terms of counting accuracy [39]. However, most existing works focus on counting a single class of objects in the images. Furthermore, the images used in most existing object size estimation works only contain one type of the object [28, 29, 7]. When there exist multiple object classes to be counted in the images, multiple models need to be trained and deployed. It is imperative to have one unified model for multi-class object counting and object size estimation. However, the lack of datasets containing multiple object classes in the images and proper datasets for object size estimation has been the main barrier to such research.

Another barrier to current research is they have not explored the most advanced deep learning techniques but use the traditional detection algorithms for size estimation. Such methods are based on hand-crafted pixel counting [13, 11, 29], which has lower performance and cannot return size and position information for the individual object instance. Unsurprisingly, how to detect and identify individual object instances for different types of objects in multi-class object size estimation is still an open problem. Existing methods such as Connected-component labelling (CCL) [36] and the Watershed algorithm [27] can only do segmenting for all objects in the image but cannot identify different types of objects.

1.2. Methodology

To overcome such barriers, we first create a benchmark dataset for multi-class object counting and size estimation by generating synthetic images using the Unity game engine [22]. We can conveniently generate a large number of images containing one or multiple classes of objects and the corresponding point annotations of the object instances with nearly zero effort. In addition, the size of each object instance in the synthesized images can also be controlled by the Unity engine which allows us to obtain the ground truth object size information easily for each object instance. As a result, the created synthetic dataset can serve as a proper testbed for both multi-class object counting and object size estimation.

We unify the multi-class object counting and size estimation in one problem and propose a solution to address this problem using deep convolutional neural network-based models. This unified problem has a variety of potential applications and a big impact in the real world. For example, an automatic cash register system [44] has the requirement of recognising all the fruit instances and estimating their sizes based on which different prices can be applied. In intelligent farm systems [34, 4, 2, 5], automatic plant and animal counting and size estimation can provide useful information for monitoring the growth of plants and livestock.

Our proposed solution is based on the commonly used framework for single-class object counting. Specifically, we use a backbone network to extract deep feature maps from the original images. The backbone network can be any state-of-the-art deep learning model for image classification as demonstrated in [42]. The backbone network is followed by two parallel heads aiming at outputting the predicted count and size information respectively. The count information is represented by a multi-class density map which is a stack of density maps for all object classes. Similarly, the size information is represented by a multi-class size map. A size map consists of the size of each object instance in the image. To overcome the barrier in current research of multi-class object size estimation, an individual object size retrieval algorithm is proposed to identify the individual object and calculate its size from the predicted size map of an image. A generic target matching algorithm is also proposed to identify true positive (TP), false positive (FP), and false-negative (FN) instances from the list of predicted individual objects for the performance evaluation. In this paper, we do not bother to estimate the true diameter, length or volume of the object but they can be derived by multiplying a constant factor (e.g., a value relating to the camera distance to the objects and the camera distance is fixed in our datasets)

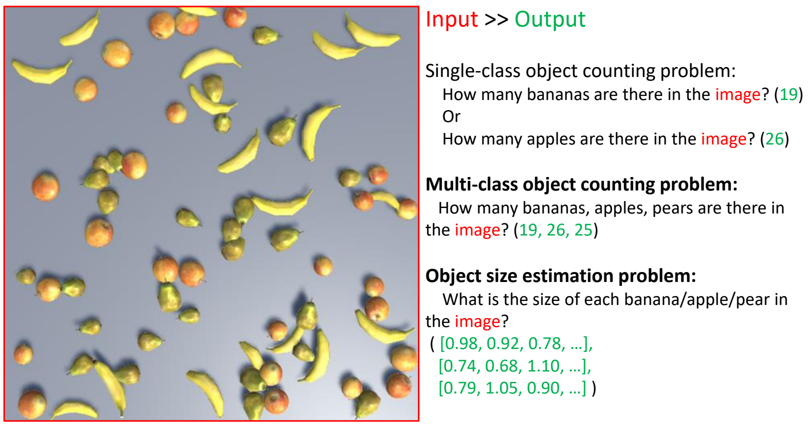

Input >> Output
Single-class object counting problem:
How many bananas are there in the image? (19)
Or
How many apples are there in the image? (26)
Multi-class object counting problem:
How many bananas, apples, pears are there in the image? (19, 26, 25)
Object size estimation problem:
What is the size of each banana/apple/pear in the image?
([0.98, 0.92, 0.78, ...],
[0.74, 0.68, 1.10, ...],
[0.79, 1.05, 0.90, ...])

Figure 1: An illustration of the unified problem of multi-class object counting and size estimation.

with the estimated sizes. In other words, we focus on the estimation of the relative scale of an object and refer to this relative scale as object size without causing ambiguity.

1.3. Contribution and paper structure

The contributions of this work can be summarised as follows:

- We formulate a unified problem of multi-class object counting and size estimation and propose an approach to solve the problem by estimating class-wise density maps and size maps.
- We create a benchmark dataset to evaluate different methods for the multi-class object counting and size estimation problem.
- Extensive experiments on the proposed dataset are conducted to validate the effectiveness of the proposed approach to the multi-class object counting and size estimation problem.

The rest of the paper is organized as follows: In Section 2, several state-of-the-art automated object counting and size estimation works are reviewed. The details of our generated dataset are introduced in Section 3. Our proposed methods for single and multi-class object counting and size estimation are depicted in Section 4. In Section 5, we illustrate the designed experiments and employed evaluation metrics. The results of our designed experiments are analysed in Section 6. Section 7 concludes the paper.

2. Related Work

In this section, we review works relating to ours in the literature. To the best of our knowledge, it is the first time the multi-class object counting and size estimation are formulated as a unified problem to address. There exist works focusing on single-class object counting [1, 35, 16, 41, 39, 16, 21, 26], multi-class object counting [12, 46] and object size estimation [28, 29, 7, 43, 3] respectively.

2.1. Single-Class Object Counting

Single-class object counting has been studied in a variety of applications where varied objects need to be counted within still images [1, 35, 16, 41]. Among them, there are a large number of works on crowd counting which aims at counting people in still images [39, 47, 50, 49].

Liu et al. [24] present a novel fruit counting pipeline that combines deep segmentation, frame-to-frame tracking, and 3D localization to accurately count visible fruits across a sequence of images. Aich et al. [1] focus on the counting of rosette leaves from an RGB image. In their method, the foreground is segmented as a first step and the segmentation results are fed into a regression model which outputs the final count of leaves. Salami et al. [35] and Tong et al. [41] aim at counting trees from aerial images. Both of them leverage the pipeline of image segmentation and counting. The former employs traditional image processing methods whilst the latter utilises convolutional neural networks for improved performance.

Another prevalent research topic of single-class object counting is crowd counting in still images or surveillance videos. Recent advances in crowd counting are mostly based on the use of deep convolutional neural networks to predict the density map. For a thorough review of state-of-the-art approaches to crowd counting, we refer the readers to the survey papers [39, 16, 21].

In contrast to these single-class object counting works, our work focuses on multi-class object counting where there exist multiple types of objects in an image and we aim to count the number of instances for each class.

2.2. Multi-Class Object Counting

There exist very few works on multi-class object counting [12, 46]. Only very recently, Xu et al. [46] attempt to address the multi-class object counting problem by adapting the traditional crowd counting frameworks to the one being able to handle multiple classes. Specifically, they extend the number of density maps in the output of the deep neural networks from one to the number of classes. This is a straightforward idea and will also be employed in our work. Moreover, the authors claim there exists mutual interference in the shared feature space among different classes. To this end, a multi-mask structure is used so that the class-wise attention maps are learned and attended to the predicted class-wise density map. This is an extension of segmentation based attention in the single-class crowd counting [42] which, however, applies the attention maps to the feature maps instead of the predicted density maps. Go et al. [12] employ a different architecture for multi-class object counting. Instead of predicting the per-class density maps directly, they predict the total density map and class-wise probability maps respectively from two parallel branches of networks. The per-class density map can be derived by the element-wise multiplication operation between the total density map and the class-wise probability maps.

Regarding the datasets for multi-class object counting, Xu et al. [46] use the VisDrone-DET [52] and RSOC [9] datasets in their experiments. The VisDrone-DET dataset was originally collected for object detection with bounding box annotations which can be easily converted into point annotations commonly used in crowd counting for more practical use. Road users including pedestrians, persons, cars, vans, buses, trucks, motors, bicycles, awning-tricycles and tricycles are considered and annotated in the dataset. The RSOC dataset was collected from Google Earth and the DOTA dataset [45] for remote sensing. There are four object classes including small vehicles, large vehicles, ships and buildings in the RSOC dataset. Go et al. [12] collected a fine-grained multi-class object counting dataset, named KR-GRUIDAE, which was claimed the first large dataset focusing on fine-grained multi-class object counting. The images of this dataset were captured by ecologists and contain the Red-Crowned Crane (RCC), Red-Crowned Crane Juvenile (RCCJ), White-Naped Crane (WNC), White-Naped Crane Juvenile (WNCJ), and Anser albifrons (AA) in various poses and views.

However, all of the aforementioned datasets share limitations in that there are only a few object classes and the numbers of object instances in the images are not well controlled during the data collection (e.g., many images contain only one class of objects, some classes only appear in a few images). As a result, they may not be suitable testbeds for the research in multi-class object counting. To fill this gap, we propose a new dataset MOCSE13 using synthetic images without exhaustive manual data annotations. We believe this new dataset will benefit the research on the topic of multi-class object counting and many others.

2.3. Object Size Estimation

Image based object size estimation can be very useful in real-world applications such as food calorie estimation [28, 7], fruit yield estimation [13, 11], etc. For food calorie estimation, there are usually a small number of food types and instances in the image and object segmentation methods are used for food size estimation. Recently, deep learning based fruit yield estimation has been widely studied for different types of fruit such as apple [13, 11], mango [43], citrus [3], strawberry [29], olive [32], etc.

Most approaches rely on the object detection results in the first step and estimate the object sizes based on the detection results [13, 11, 29]. In our work, we do not explicitly detect each object instance, instead, we directly regress the object sizes from the raw images. [13] uses a machine vision system consisting of a colour CCD camera and a time-of-flight (TOF) light-based 3D camera for estimating apple size in tree canopies. The approach in [11] generates 3D point clouds to model the detected apples for size estimation. In contrast, our work does not focus on the precise volume of the objects but approximates the relative size of individual objects within the same image. In-field mango fruit size estimation method proposed in [43] focuses on the estimation of fruit-to-camera distance using an RGB-D camera and the mango sizes in the pixel space. Again, a detection algorithm was used based on the hand-crafted image features. We argue that deep learning methods should be used to replace the traditional detection methods for improved performance.

Deep learning based methods are used in [3] for fruit size estimation. Specifically, a Faster-R-CNN model [33] was employed to detect the citrus fruit objects in images captured by a UVA from the top view. However, object detection methods require bounding box annotations for training which are far more costly than point annotations required by our proposed solutions. Our work also focuses on the object size estimation within images captured from the top view but the images are captured in more controlled conditions (the number of object instances and object types can be properly set in Unity for generating synthetic images) and the approaches developed based on our synthetic data can be applied to real-world scenarios such as images captured by a UVA from the top view.

In summary, most existing object size estimation methods in the literature have the limitations: 1) only one type of object exists in the images; 2) most of them have not explored the most advanced deep learning techniques but use the traditional detection algorithms followed by hand-crafted pixel counting methods for size estimation. In our work, we aim to investigate the effectiveness of state-of-the-art deep learning models for object size estimation.

3. MOCSE13: A Synthetic Dataset

The images of the dataset are generated from the Unity engine. We first outline the process of image generation using Unity and subsequently, the technical details are given.

- First randomly select an object from a set of 3D models then generate it above a panel in Unity with random sizes.
- Repeat the last step a certain number of times where this number is randomly selected from a pre-defined number range.
- All generated objects will fall to the panel while the Unity engine simulates their physical effects during the falling process.
- The camera in Unity screenshots the panel after all objects are stable and then outputs it along with the object information file.
- Destroy all objects to clear the panel and then start to generate the next image.

The original 3D models of 13 classes of fruit or vegetable are obtained from an Asset ”Supermarket with LOD” which was derived from Unity Asset Store [8]. The 3D model defines the framework and the attached material is the surface of the object lookalike in the Unity scene. The material can be changed to the picture taken from the real fruits or vegetables. This will make sure the diversity of the objects in the dataset. In Unity, we set the physics properties for all 3D objects to simulate the physical effect in the real world and the collision effect to collide with other objects. After these settings, these 3D objects were saved as Prefabs. The Prefab in Unity allows the user to create, configure, and store an object with all its components, property values, and child object as a reusable asset. The Prefab Asset acts as a template when we create new Prefab instances in the scene of Unity.

At the start of the dataset generating process, the fruit or vegetable object is randomly selected from a set of Prefabs and then instances of random sizes are generated. This set of Prefabs only contains one class of object Prefabs when generating the single-class dataset. By contrast, such set of Prefabs will contain all selected classes of object Prefabs when generating the multi-classes dataset. The number of the generated objects is randomly selected from a pre-defined number range. Under the gravity simulation, all generated objects fall to the panel. With the Rigidbody component, each object's position and rotation are controlled through the physical simulation. The Mesh Collider component enables the object to collide with other objects. Once all objects are stable on the panel, the camera above the panel takes a screenshot with a pre-set resolution. The corresponding data of all objects such as position, size, and class label are saved to a text file. This information could mark objects in the image precisely. After saving the image and text file, all the objects on the panel are destroyed. The next image will then start to generate. Using different random seeds, an infinite number of images can be generated.

Our created benchmark dataset for Single-class object counting has the following settings:

- 30 images per class, and the resolution of the image is 512*512.
- 13 classes of fruits or vegetables: apple, artichoke, avocado, banana, carrot, courgette, garlic, melon, onion, orange, pear, tomato, and shallot. Figure 2 shows the example Prefabs of these classes as appeared in Unity.
- 6 groups which are the combination of counts in different ranges (large, medium, and small amount of instances) and size in different ranges (large, medium and small size of instances) and each group has 5 images. Table 1 gives the detail of all six groups.
- A text file is provided for each image in the dataset which contains the ground truth information of the corresponding image. The ground truth information includes the total number of instances; label, size and position of each instance.

As said before, the set of Prefabs which were used to generate instances in the image contains all selected classes of object Prefabs when generating a multi-class dataset. Therefore, the total number of all objects in the images is the sum of the number of instances of each class. This is the only difference between the Single-class dataset and the Multi-class dataset. Figure 3 shows examples of ground truth density maps and density size maps, and corresponding estimated density map and size map for one class (avocado) of an image in the Multi-classes dataset.

Our created benchmark Multi-class datasets have the following settings:

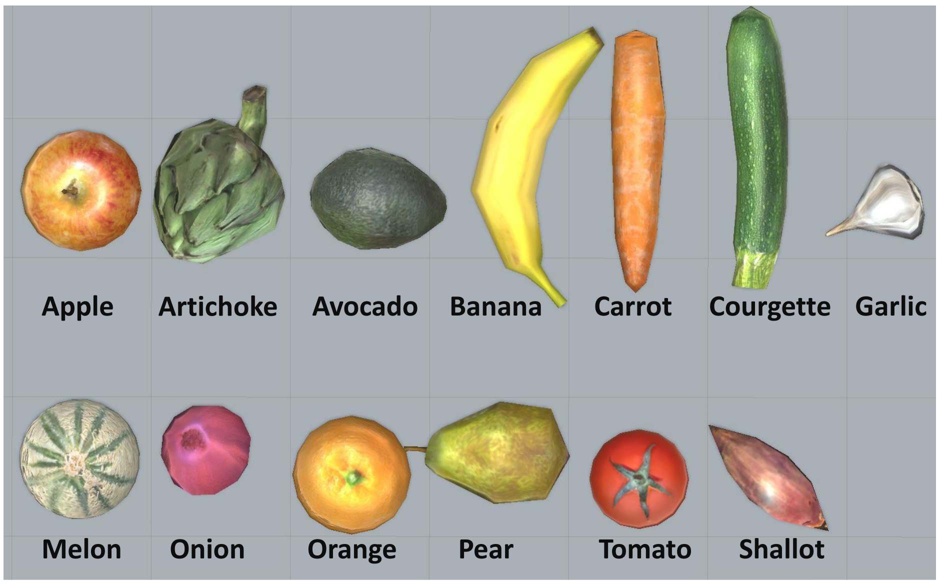


Figure 2: The example Prefabs of 13 classes used in Unity to generate our created benchmark datasets.

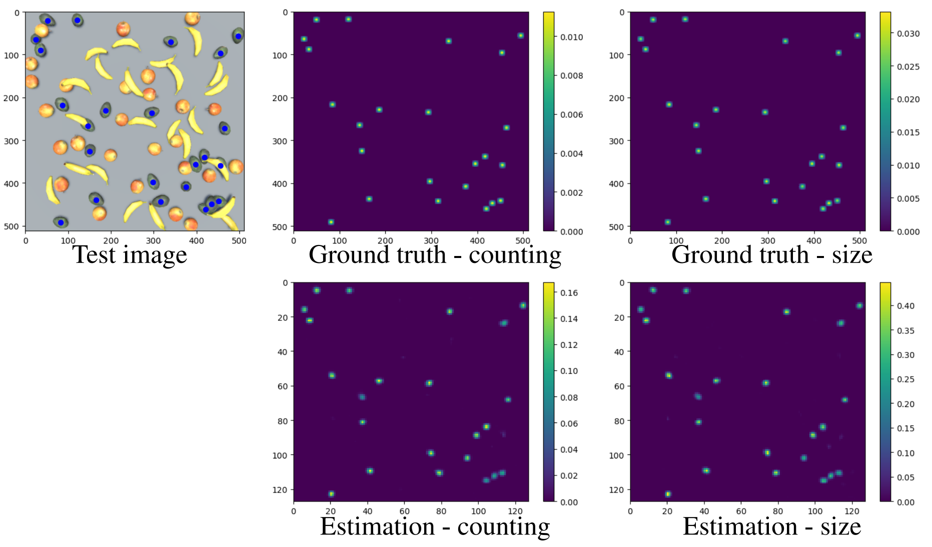


Figure 3: The ground truth density and size maps and estimated density and size maps of our model for the class of avocado.

Table 1: Detail of six groups of images in the dataset.

	Small amount	Large amount
Small size	Number range: 30-70; Size range: 0.6-0.8	Number range: 70-110; Size range: 0.6-0.8
Medium size	Number range: 30-70; Size range: 0.8-1.0	Number range: 70-110; Size range: 0.8-1.0
Large size	Number range: 30-70; Size range: 1.0-1.2	Number range: 70-110; Size range: 1.0-1.2

- 4 example Multi-class datasets are generated, and there are 3 classes in the images of each dataset.
- 30 images per dataset and the resolution of the image is 512*512.
- 6 groups which are the combination of counts in different ranges (large, medium, and small amounts of instances) and sizes in different ranges (large, medium, and small sizes of instances) where each group has 5 images. Table 1 gives the detail of all six groups. This count number is the total number of all objects in the image.
- A text file is provided for each image in the dataset which contains the ground truth information of the corresponding image. The ground truth information includes the total number of instances; label, size and position of each instance.

Table 2 summarizes a comparative analysis of state-of-art object counting and size estimation datasets with our generated dataset MOCSE13. From the table, we could see the following facts. First, all the existing object counting datasets are the single-class counting dataset which means only one type of object is annotated in the image. Besides, most of them focus on the crowd counting such as ShanghaiTech [51], UCF_QNRF [19], UCF_CC_50 [18]. Although some of the datasets have more images compared with ours, they are hand-annotated which could be very costly and time-consuming. Furthermore, different from single-class object counting, our work focuses on multi-class object counting where there exist multiple types of objects in an image and we aim to count the number of instances for each class. Secondly, there is no public available multi-class object counting dataset. Some works such as Xu et al. [46] can only utilize the single-class object counting dataset which includes multiple subsets and the object detection dataset to evaluate their work of multi-class object counting. Even after the pre-processing, such datasets still only contain a few object classes and the numbers of object instances in the images are not well controlled during the data collection. All these imply a suitable dataset is urgently

Table 2: Comparison between current single/multi-class object counting and size estimation datasets and MOCSE13.

Dataset	No. of images	No. of classes	Public availability	Object counting	Size estimation	Extensible	Mark method
ShanghaiTech (A)[51]	482	1	✓	✓	×	×	hand-annotated
ShanghaiTech (B)[51]	716	1	✓	✓	×	×	hand-annotated
UCF_QNRF[19]	1535	1	✓	✓	×	×	hand-annotated
UCF_CC_50[18]	50	1	✓	✓	×	×	hand-annotated
Arabidopsis [1]	810	1	✓	✓	×	×	hand-annotated
KR-GRUIDAE [12]	1423	5	✓	✓	×	×	hand-annotated
Starberry [29]	337	1	×	×	✓	×	hand-annotated
Boiled-rice [7]	360	1	×	×	✓	×	hand-annotated
UAV-Orange [3]	300	1	×	×	✓	×	hand-annotated
MOCSE13 (ours)	510	**13**	✓	✓	✓	✓	**auto-annotated**

required as suitable testbeds for multi-class object counting research. Thirdly, there is only one item in the image of most of the existing object size object estimation datasets, and no existing work focuses on multi-class object size estimation.

Here we list the following advantages of our generated dataset MOCSE13 to show how we fill the gaps in currently object counting and size estimation datasets identified by Table 2:

- All the items in the images of dataset MOCSE13 are automatically annotated when the images are generated, which is efficient and time-saving.
- All the generated images are randomly generated and non-repeated. Although our dataset is a generated synthetic dataset, we can generate new images conveniently based on the needs and requirements. This means our dataset has advantages in the aspects of dataset size and variety compared with other datasets.
- Based on the transfer learning, our dataset could be used to train a deep learning model and this model can be directly used in the real applications of object counting and size estimation without further model training or fine-tuning [6, 37]. Useful prior knowledge can be gained by deep learning and pre-training on our dataset. Afterwards, the model is equipped with pre-trained weights for the object counting or size estimation and can be easily adapted in the real application with further training or fine-tuning.

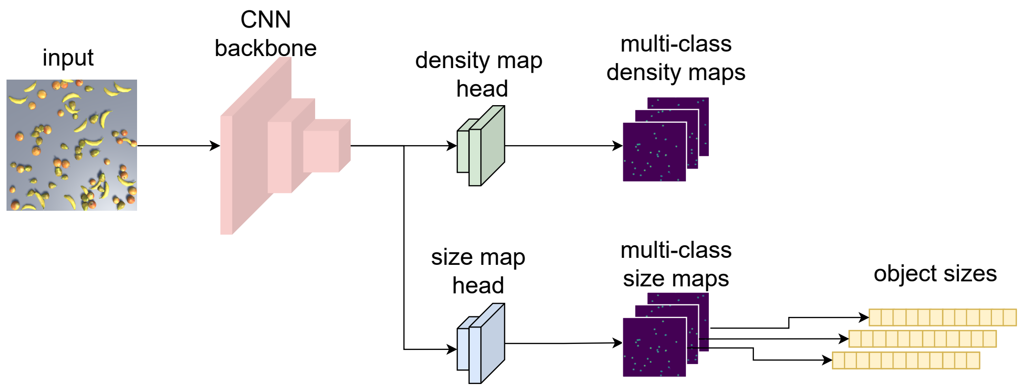


Figure 4: The framework of our proposed approach to multi-class object counting and size estimation.

4. Method

In this section, we introduce our proposed approach to unified multi-class object counting and object size estimation. The extension of traditional object counting to multi-class scenarios is first presented. Subsequently, we propose a novel method for multi-class object size estimation. The framework of our proposed approach is illustrated in Figure 4. Our approach is implemented in PyTorch [31] and source code is publicly available[1].

4.1. Multi-Class Object Counting

Most approaches for single-class object counting are based on a similar framework in which the raw image is fed into a deep Convolutional Neural Network (CNN) and the output is a density map. Since the idea of the density map was first proposed in [23], density estimation-based methods become the most common way to solve the problem of object counting in images. The use of the deep CNN model in large-scale datasets to estimate the density map could further improve the accuracy of object counting. The motivations for the use of deep CNN are usually to improve the generalization to scale-variant object images [42]. The backbone of the deep CNN model can be any state-of-the-art models for image classification such as VGG [38], Inception-V3 [40], DenseNet [17], ResNet [15], TEDNet [20], etc. The choice of different backbones depends on the trade-off between computational cost and accuracy as demonstrated in [42]. The

[1]https://github.com/liulei1260/MOCSE13

final object count can be derived from the output density map by simply aggregating all the pixel values in the density map. This is based on the fact of how the density map is defined.

For density maps $\boldsymbol{M}^{den} \in \mathbb{R}^{+H \times W}$, where H and W are the height and width of the image, we follow [42] using a Gaussian kernel $G_\sigma \in \mathbb{R}^{+15 \times 15}$ with a kernel size of 15×15 and fixed $\sigma = 4$:

$$\boldsymbol{M}^{den}(\boldsymbol{x}) = H(\boldsymbol{x}) * G_\sigma(\boldsymbol{x}), \tag{1}$$

where $H(\boldsymbol{x}) = \sum_{i=1}^{N} \delta(\boldsymbol{x} - \boldsymbol{x}_i)$, N is the number of point annotations in the image and $\delta(\cdot)$ is the Delta function. As a result, $H(\boldsymbol{x})$ is a binary matrix of the same size as the image and only has values of ones at the positions of point annotations. The density map is derived by the convolution between $H(\boldsymbol{x})$ and the Gaussian kernel $G_\sigma(\boldsymbol{x})$.

To adapt the traditional single-class object counting (e.g., crowd counting) methods to the multi-class object counting problems, we modify the density map head to output multiple density maps whilst keeping the backbone of the deep model for feature representation learning. For the c-th class, the ground truth density map is computed by:

$$\boldsymbol{M}_c^{den}(\boldsymbol{x}) = H_c(\boldsymbol{x}) * G_\sigma(\boldsymbol{x}), \quad c = 1, 2, ..., C. \tag{2}$$

As a result, the multi-class density map $\boldsymbol{M}^{den} \in \mathbb{R}^{+C \times H \times W}$ is a stack of per-class density map, where C is the number of classes.

4.2. *Object Size Estimation*

Our proposed approach to object size estimation is composed of two stages. In the first stage, per-class object size maps are generated by the size map head which follows the backbone network as shown in Figure 4. In the second stage, we try to retrieve the size of each object from the predicted size map. To this end, we propose a method for individual object size retrieval (c.f. Section 4.2.2).

4.2.1. *Object size map prediction*

Inspired by the density map prediction, we attempt to predict the object size map for object size estimation. In the density map, an object is represented by a Gaussian kernel in the local region where the object is located in the image. Similarly, we represent the size of an object by a Gaussian kernel weighted by the corresponding object size and the Gaussian kernel is located in the local region of the object. As a result, the object size maps can be predicted in the same way as the density maps. As shown in Figure 4, a size head is attached to the backbone network to transform the feature maps into the predicted object size maps.

Following the notations used in Section 4.1, the ground truth multi-class size maps can be computed by:

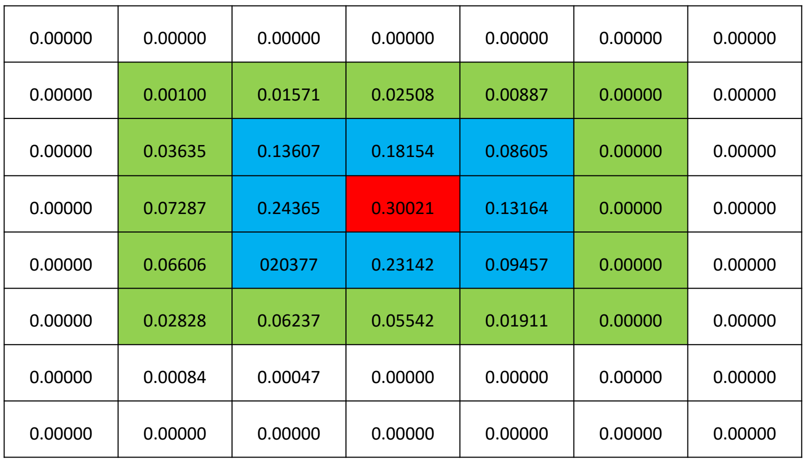
$$\boldsymbol{M}_c^{size}(\boldsymbol{x}) = S_c(\boldsymbol{x}) * G_\sigma(\boldsymbol{x}), \quad c = 1, 2, ..., C. \tag{3}$$

where $S(\boldsymbol{x}) = \sum_{i=1}^{N} s_i \cdot \delta(\boldsymbol{x} - \boldsymbol{x}_i)$, N is the number of point annotations in the image and $\delta(\cdot)$ is the Delta function. By comparison with the $H_c(\boldsymbol{x})$ in Eq.(2), $S(\boldsymbol{x})$ is also a matrix of the same size as the image but has values of object sizes (i.e. s_i) at the positions of point annotations. The size map is derived by the convolution between $S_c(\boldsymbol{x})$ and the Gaussian kernel $G_\sigma(\boldsymbol{x})$. We use the same Gaussian kernel in Eq.(2) and Eq.(3).

4.2.2. Individual object size retrieval

In this subsection, the method for individual object size retrieval in our approach is introduced. There are two main purposes of this method: to identify the individual object and to calculate its size from the predicted size map of a text image.

Without the ground truth, identifying the individual object in the size map is the first step of our retrieval method. As explained before, the size map is patched by the Gaussian kernels. The Gaussian kernel is an isotropic function and the weight decreases as moves away from the centre. The weight of the centre element is the maximum compared with its surrounding elements. Based on this feature, we could identify all individual objects from the size map by a loop through elements in the size map. Figure 5 shows part of an example size map, the weight in each box is calculated by Eq.(3). The element in the red box has the maximum weight compared with the weight of those elements in the blue boxes surrounding it. This element is then marked as the centre of an object. To get the predicted size of this identified object, we need to add all the weight of all elements in the red, blue, and green boxes. All these boxes are combined into a 5×5 kernel. Since the size map is a size reduction version of the test image after several convolutions, there exists a shrinking ratio set in the implemented backbone CNN model. For example, we set this ratio as 4 for “Inception-V3”, and “VGG-16-BN”. The patched Gaussian kernel represents a predicted object and its size is set as 15×15 in our approach. Therefore, this 5×5 kernel in the size map covers almost all the elements of the patched Gaussian kernel after several convolutions with a shrinking ratio of 4. Note this kernel size may be different with



0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.00000	0.00100	0.01571	0.02508	0.00887	0.00000	0.00000
0.00000	0.03635	0.13607	0.18154	0.08605	0.00000	0.00000
0.00000	0.07287	0.24365	0.30021	0.13164	0.00000	0.00000
0.00000	0.06606	020377	0.23142	0.09457	0.00000	0.00000
0.00000	0.02828	0.06237	0.05542	0.01911	0.00000	0.00000
0.00000	0.00084	0.00047	0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Figure 5: The example of a partly size map (matrix style): red box is the centre element of a Gaussian kernel, blue boxes are used to check the centre element of the kernel and green boxes are the edge of the kernel.

different backbone CNN models due to the different shrinking ratio settings. The weight of the elements which is not included in this kernel is relatively small and therefore could be neglected. After this search process, a list of identified individual objects along with their position and size information are obtained.

The next step is to identify the true positive (TP), false positive (FP), and false-negative (FN) instances from the list of predicted individual objects. The basic idea is to match the predicted object with an object from the list of ground truth objects which has the minimum distance with it in the size map. We consider two scenarios when matching the objects from two lists: the length of the predicted objects list (POL) is greater than the ground truth objects list (GTOL), and the opposite scenario. In both cases, the positions of the predicted objects are multiplied by the shrinking ratio first. The next step is to match the closed pairs of objects from two lists based on Eq.(4), where D_{AB} is the distance between objects A and B, and y_A and x_A represent the position of the object A.

$$D_{AB} = \sqrt{(y_A - y_B)^2 + (x_A - x_B)^2} \tag{4}$$

In the first scenario, for each object in GTOL, we find an object from the POL which has a minimum distance with it in the size map and then mark this pair of objects with a match. Sometimes, there may exist some objects in the POL which involve multiple matches. These duplicate matches are caused by the overlapping items in the test figure. To identify the correct match, we compare the size differences among these matched pairs. The object in the GTOL which has the minimum size difference with this popular matched object in the POL keeps the match. In the second scenario, for each object in POL, we match it with an object from the GTOL which has a minimum distance from it in the size map. Also, we use the same way to identify the correct match for the objects in the GTOL which has multiple matches. The multiple matches in this scenario may be caused by the wrong prediction.

After the matching process, the objects in the POL which are not matched are marked as FP; the objects in the GTOL which are not matched are marked as FN and the matched objects in the two lists are marked as TP. Then we can calculate the metrics of precision, recall, F1-score, and MAPE for the prediction. The detail of these evaluation metrics is presented in Section 5.1. The first step of our retrieval method -identifying the individual object in the size map- is presented in Algorithm 1: Individual Object Size Retrieval (IDSR) algorithm. Note, the s in row 6 is the size of the kernel which is related to the shrinking ratio set in the backbone CNN model. We set s as the ceiling value of the ratio between 15 and the size ratio, then add one to cover all the elements of the patched Gaussian kernel (size 15×15). The algorithm of how to identify the TP, FP, and FN instances from the list of predicted individual objects is shown in Algorithm 2: Target Matching (TM). Since our approach outputs the size map for each class separately in the multi-class size problem, the proposed Algorithm 1 and 2 can also be used to do the individual object size retrieval for the multi-class size problem.

To analyze the time complexity of Algorithm 1, the number of elementary operations of each step needs to be counted. Assuming both the length and the width of a size map are n, where the n equals 512 in our case. The purpose of the first part of Algorithm 1 is to identify individual objects from the size Map. We use a sliding window in size map to loop all the points. For each point, it will compare with all of its surrounding points once at most. If the object is identified at this point, 3 further operations will be performed to get the related information. Therefore, the time complexity of this algorithm is $O(n^2)$. The Algorithm 2 focuses on the operations of lists of GTOL and POL. Assume the number of the object in this size map is m. That means the length of GTOL or POL is m or close to m. To match the objects between two lists which needs at most

$(m \times m)$ operations and the related time complexity is $O(m^2)$. The rest code in the second part is dealing the multiple matches for each object in the lists and the time complexity is also $O(m^2)$. Hence, the time complexity of Algorithm 2 is $O(m^2)$. The maximum units of memory space required to run Algorithm 1 and Algorithm 2 are $(n^2 + m^2 \times 2)$ that are used to save the size map and two lists. In conclusion, the time complexity of Algorithm 1 and Algorithm 2 are $O(n^2)$ and $O(m^2)$, repetitively. This means the execution time of our proposed algorithm is either given by a polynomial on the size/time of the input or can be bounded by such a polynomial.

Algorithm 1 Individual Object Size Retrieval

```
Input: The size map and the ground truth object list (GTOL),
Output : The the predicted objects list (POL).

1: procedure IDENTIFY INDIVIDUAL OBJECTS FROM THE SIZE MAP
2:     for each point in the size map (matrix) do
3:         if the value of current point > all surrounding points then
4:             Current point ← the centre of an identified object's Gaussian kernel.
5:             The coordination of this object ← current position.
6:             The size of this object ← sum of all values of $s \times s$ kernel around current point.
7:             Append this object and its information to the POL.
8:         end if
9:     end for
10: end procedure
```

5. Experiments

5.1. Evaluation Metrics

In this section, we describe the evaluation metrics used for object counting and object size estimation in our experiments. For single-object counting, we follow the previous works [42] and use three metrics, i.e., the mean absolute error (MAE), the root mean squared error (RMSE) and the mean absolute percentage error (MAPE) to evaluate the performance. The three metrics

Algorithm 2 Target Matching

Input: The predicted objects list (POL) and the ground truth object list (GTOL),

Output : Identified TP, FP and FN.

```
1:  procedure IDENTIFY THE TP, FP AND FN INSTANCES
2:      if the length of POL $>$ the length of GTOL then
3:          for each object in the GTOL (list) do
4:              match currently object with a closed object from POL in the size map.
5:          end for
6:      else if the length of POL $<$ the length of GTOL then
7:          for each object in the POL (list) do
8:              match currently object with a closed object from GTOL in the size map.
9:          end for
10:     end if
11:     if exists an object in the POL or GTOL which has multiple matches then
12:         Identify the correct match by size difference of each matched pair;
13:         Keep the match for the pair with minimum size difference.
14:     end if
15:     FP, FN $\leftarrow$ the objects in the POL and GTOL without match, respectively.
16:     TP $\leftarrow$ the matched objects in the two list.
17: end procedure
```

for a specific class c can be calculated as follows:

$$MAE_{count}^{c} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} |y_i^c - \hat{y}_i^c| \tag{5}$$

$$RMSE_{count}^{c} = \sqrt{\frac{1}{N_{test}} \sum_{i=1}^{N_{test}} (y_i^c - \hat{y}_i^c)^2} \tag{6}$$

$$MAPE_{count}^{c} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{|y_i^c - \hat{y}_i^c|}{y_i^c} \tag{7}$$

where y_i^c and $\hat{y}_i^c$ are the ground truth and predicted count for the c-th object class in i-th test image respectively, N_{test} is the number of test images.

For multi-class object counting, we calculate the per-class evaluation metrics (e.g., MAE_{count}^{c}, $RMSE_{count}^{c}$ and $MAPE_{count}^{c}$) for each class using Eqs.(5-7) and the averages over all C classes can be calculated as:

$$MAE_{count} = \frac{1}{C} \sum_{c=1}^{C} MAE_{count}^{c} \tag{8}$$

$$RMSE_{count} = \frac{1}{C} \sum_{c=1}^{C} RMSE_{count}^{c} \tag{9}$$

$$MAPE_{count} = \frac{1}{C} \sum_{c=1}^{C} MAPE_{count}^{c} \tag{10}$$

For the object size estimation, we first evaluate the performance of individual object retrieval using the metrics of precision, recall and F1-score which can be computed as follows:

$$Precision = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{TP_i}{TP_i + FP_i} \tag{11}$$

$$Recall = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{TP_i}{TP_i + FN_i} \tag{12}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{13}$$

where TP_i, FP_i, FN_i are the numbers of true positive, false positive and false negative instances in the i-th test image, respectively.

Based on the retrieval results, we evaluate the size estimation errors for each object class using MAPE. We do not use MAE and RMSE for the evaluation of object size estimation since we do not care about the absolute values of object sizes in our experimental settings. The MAPE for object size estimation is calculated over all the object instances in all the test images for a specific object class c.

$$MAPE^{c}_{size} = \frac{1}{\sum_{i=1}^{N_{test}} n_i} \sum_{i=1}^{N_{test}} \sum_{j=1}^{n_i} \frac{|s^{c}_{ij} - \hat{s}^{c}_{ij}|}{s^{c}_{ij}} \tag{14}$$

where s^{c}_{ij} and $\hat{s}^{c}_{ij}$ are the ground truth and predicted sizes of the j-th object instance belonging to the c-th class in the i-th test image. When there are multiple object classes in the test images, the overall performance can be evaluated by the average of MAPE overall C classes:

$$MAPE_{size} = \frac{1}{C} \sum_{c=1}^{C} MAPE^{c}_{size} \tag{15}$$

5.2. Experimental settings

In this section, we illustrate the experimental settings and designs of our all experiments.

5.2.1. Experiment of single-class object counting and size estimation

For the experiment of single-class object counting and size estimation (Section 6.1) VGG-16-BN is implemented in PyTorch [31]. The "Adam" optimizer is employed for training. This experiment aims to evaluate the approach to object counting and size estimation for the single-class object. The learning rate is set to 1e-4 and reduced by a factor of 0.5 after every 50 epochs. Since the model can always converge after around 100 epochs for all test classes, we set the total number of the training epochs to 200. As stated in Section 3, we have 30 images with labels of class and size for each class. There are 13 classes of fruit or vegetable in total. Each class of dataset contains 6 groups which is the combination of counts in different ranges and sizes. In the experiments, we randomly select three images from each group as the training image set and the rest of the two images are set as the test image set. The training and testing ratio is 1.5 in our case. During the training process, rather than preparing the patch in advance, we randomly select 8 images from the training set and 4 patches are randomly cropped from each selected image. Therefore, each iteration of training generates a batch of 32 training patches. In this way, the model processes more diverse training patches, which helps to alleviate the potential over-fitting problem. The output of VGG-16-BN has the size of $128 * 128$ (i.e. $1/4$ of the input size). The method we use to adapt the ground truth density and size map is the sum-pool, which has the same size as the output. As stated in [14] [48], horizontally flipping the training patches and corresponding density or size map with a set probability for data augmentation could be beneficial for the training process. Therefore, we set this probability as 0.5. For testing, the density map or size map of the test figure could be output by feeding the whole image to the network. The predicted count can be computed by aggregating all the pixel values in the density map. The size map is used for the individual object size retrieval process.

For the single-class object counting and size estimation, we also design and implement a separate experiment to examine the effect of different number of the class instance in the training and test images on the prediction result. Except for the training and test sets of images, the rest of the experiment setting remains the same as the experiment of single-class object counting and size estimation. Based on Table 1, we create three comparative experiment datasets. We first use the images which contain the large and medium size of instances to train the model and images contain the small size of instances for testing; then use the images which contain the small and large size of instances to train the model and images contain the medium size of instances to test; finally, use the images which contains the small and medium size of instances to train the model and images contain the large size of instances to test.

5.2.2. Experiment of multi-class object counting and size estimation

For the multi-class object counting and size estimation, we design 4 comparative experiments. In each experiment, 3 classes of fruits or vegetables are selected and then we generate the training and test image dataset which only contains these 3 classes of instances from Unity. These experiment datasets also follow the setting stated in Table 1. The rest of the experiment settings keeps the same with the experiment of single-class object counting and size estimation. The results of this part are stated in Section 6.2.

5.2.3. Experiment on the different number of training images

We also evaluate the effect of the different number of training images on the prediction result in our approach. We run 5 experiments in which 6, 18, 30, 42, and 60 images are used for training, respectively. The number of the test images keeps the same as in previous experiments: 12. These datasets only contain one class of instance: *banana*, and the number of images of all 6 groups (group detail is shown in Table 1) are equally generated and selected. The corresponding experiment results are stated in Section 6.3.

5.2.4. Experiment on the different backbone networks and segmentation algorithms

The last part of the experiment is to test the different backbones of the deep CNN model. Except for the mentioned VGG-16-BN, several state-of-art models for image classification are implemented and tested which include: Inception-V3, TEDNet [20], ResNet-50, DenseNet-121 and ShuffleNet v2x0.5 [25]. The dataset used in these experiments is 30 images of the carrot dataset which was also used in the experiment of single-class object counting and size estimation. It is noteworthy that the generated density and size maps have different sizes for those models (e.g., the size ratio between input and output is 1 for “TEDNet”, 4 for “Inception-V3”, “ShuffleNet v2x0.5” and “VGG-16-BN”, 8 for “ResNet-50” and “DenseNet-121”).

Since our object size retrieval algorithm is based on the property of the patched Gaussian kernel in the predicted size map, the algorithm is not valid in the following two cases: the weights in the patched kernel are not Gaussian distributed; the patched Gaussian kernel is not in a regular shape such as squares. In the first case, detecting the location of individual objects could be a problem. For both cases, segmenting the edge of the detected objects if they touch each other or cover the neighbours is difficult (i.e. difficult to identify the individual objects and get their size). The Connected-component labelling (CCL) algorithm [36] and Watershed algorithm [27] could segment the edge of the detected objects. However, they do not have the capability of matching the segmented individual objects with the ground truth. By contrast, Algorithm 2 is applied in our study to identify the TP, FP, and FN instances from the list of segmented individual objects. Therefore, we design the experiment to prove the precision and feasibility of our proposed Algorithm 1 and Algorithm 2 for the individual object size retrieval problem. Also, this experiment aims to identify the feasible backbones of the deep CNN model for our problem. In this part of the experiment, MCNN [51], TEDNet [20] and VGG-16-BN [38] are implemented with the *apple* and *carrot* dataset. MCNN is a three-column CNN and TEDNet employs the basic *Inception* model. The experiment settings for VGG-16-BN remain the same as the experiment of single-class object counting and size estimation. The learning rate in MCNN and TEDNet is set to 1e-3 and reduced by a factor of 0.5 after every 50 epochs. The size ratio between input and output is 1 for “TEDNet”, and 4 for “MCNN” respectively. The rest of the experiment settings for MCNN and TEDNet are the same as the VGG-16-BN. To compare with the results of Algorithm 1 for the individual object detection, CCL and Watershed algorithms are implemented to process the size map obtained by MCNN, VGG-16-BN, and TEDNet. Algorithm 2 is applied to identify the TP, FP, and FN instances from the results of these three methods. To evaluate the object detection result of these methods, the object detection rate is used, which is the ratio of the number of detected objects in the predicted size map and the ground truth of the number of

Table 3: Results of single-class object counting and size estimation.

Class	Counting ↓			Retrieval ↑			Size estimation ↓
	MAE_{count}	$RMSE_{count}$	$MAPE_{count}(\%)$	$Precision(\%)$	$Recall(\%)$	$F1(\%)$	$MAPE_{size}(\%)$
Tomato	0.67 ± 0.34	0.93 ± 0.47	0.96 ± 0.47	99.78 ± 0.11	98.03 ± 0.55	98.89 ± 0.24	5.16 ± 0.46
Orange	0.89 ± 0.12	1.12 ± 0.13	1.31 ± 0.12	99.06 ± 0.53	98.45 ± 0.46	98.74 ± 0.36	5.47 ± 0.14
Avocado	1.19 ± 0.14	1.71 ± 0.16	1.61 ± 0.20	98.17 ± 0.90	97.78 ± 0.50	97.94 ± 0.69	8.54 ± 1.17
Onion	1.36 ± 0.41	1.68 ± 0.46	2.01 ± 0.57	99.26 ± 0.53	97.78 ± 0.71	98.49 ± 0.52	9.72 ± 0.79
Pear	1.49 ± 0.29	1.86 ± 0.41	2.21 ± 0.26	98.69 ± 0.54	97.38 ± 0.22	98.02 ± 0.38	9.77 ± 0.29
Melon	1.61 ± 0.62	2.03 ± 0.81	2.40 ± 1.05	99.43 ± 0.34	97.71 ± 0.25	98.55 ± 0.15	5.87 ± 0.14
Shallot	1.62 ± 0.28	2.23 ± 0.38	2.16 ± 0.45	98.97 ± 0.22	96.62 ± 0.50	97.77 ± 0.23	8.79 ± 0.50
Artichoke	1.98 ± 0.34	2.40 ± 0.28	2.87 ± 0.42	95.50 ± 1.59	98.32 ± 0.26	96.81 ± 0.88	10.92 ± 0.33
Garlic	1.98 ± 0.41	2.45 ± 0.51	3.17 ± 0.67	98.82 ± 0.44	96.61 ± 0.16	97.68 ± 0.18	11.12 ± 0.25
Apple	2.18 ± 1.06	2.86 ± 1.51	3.29 ± 1.60	97.10 ± 0.78	98.72 ± 0.35	97.87 ± 0.58	8.88 ± 0.56
Banana	3.41 ± 0.48	4.55 ± 0.63	5.11 ± 0.93	86.36 ± 1.72	95.43 ± 1.14	90.18 ± 0.73	20.51 ± 3.64
Courgette	4.53 ± 0.63	5.87 ± 0.57	6.27 ± 0.81	84.28 ± 3.30	94.17 ± 0.51	88.37 ± 1.86	18.51 ± 0.92
Carrot	5.04 ± 0.71	6.11 ± 0.81	7.43 ± 0.88	85.14 ± 2.22	94.95 ± 0.44	89.28 ± 1.34	15.49 ± 1.30

objects in the original image. The experiment results of this part are stated in Section 6.4.

6. Results

In this section, we present and analyze the results of our designed experiments. Most of the results are ordered lists in the table, from good performance to lower preference.

6.1. Results of single-class object counting and size estimation

The experiment results of single-class object counting and size estimation are listed in Table 3. From the table, we can see our model has the competitive preference on the class of *tomato* and *orange* among all 13 classes, especially *tomato* has the best $MAE_{counting}$ of 0.67 and the best $MAPE_{size}$ of 5.16. This means the instances which have the sphere shape are relatively easier to train and predict in the counting and size estimation problem. Also, the colours of these two classes are bright which could also affect the prediction performance. For the classes that have a shape of the cylinder such as *banana*, *courgette*, and *carrot*, the prediction results are relatively worse compared with other classes. The model achieves the highest $MAPE_{size}$ of 20.51 on the class of *banana* and the highest $MAE_{counting}$ of 5.04 on the class of *carrot*. The performance of the rest of the classes which have an approximate ellipse shape is intermediate. Compared with other shapes, the class of objects with sphere shape has less chance of overlapping and

Table 4: Results of single-class (apple) counting and size estimation on size-divided datasets.

Dataset detail		Counting ↓			Retrieval ↑			Size estimation ↓
Train	Test	MAE_{count}	$RMSE_{count}$	$MAPE_{count}(\%)$	$Precision(\%)$	$Recall(\%)$	$F1(\%)$	$MAPE_{size}(\%)$
Medium & Large	Small	0.89 ± 0.09	1.02 ± 0.09	1.42 ± 0.20	98.64 ± 0.73	97.80 ± 0.36	98.20 ± 0.45	9.90 ± 0.22
Small & Large	Medium	1.85 ± 0.23	2.26 ± 0.27	2.71 ± 0.15	98.50 ± 0.65	98.12 ± 0.44	98.30 ± 0.55	9.87 ± 0.75
Small & Medium	Large	2.02 ± 0.72	2.58 ± 0.91	3.06 ± 1.21	97.50 ± 0.62	98.53 ± 0.40	97.99 ± 0.36	8.40 ± 0.43

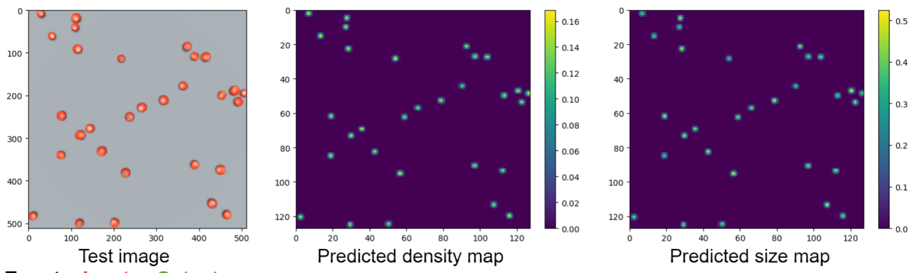
touching objects in the test image. This is the reason why the class of sphere objects such as *tomato* and *orange* has a better performance in the experiment. Figure 6 shows the example output results of tomato (best performance) and carrot (worst performance) for one of their test images respectively.

We choose *apple* to generate the size-divided dataset since its counting and size estimation results show an intermediate performance in the last experiment. This provides the space to show the effect of the size-divided dataset on the prediction performance. The running results are shown in Table 4. Using image sets of **medium** and **large** size instances for training and to image sets of **small** size instances for testing achieves the best performance on counting, $MAE_{counting}$ of 0.89; but obtains the highest $MAPE_{size}$ of 9.90 on size estimation. On the contrary, using image sets of **medium** and **small** size instances for training and image sets of **large** size instances for testing achieves the best performance on size estimation, $MAE_{counting}$: 8.40. Also, this dataset performs the highest $MAE_{counting}$ of 2.02 on counting. The results show that the **large** instances are slightly better to use for training on counting and **small** instances slightly better to use for training on size estimation. The reason is obvious, that the larger object is easier to detect and the smaller object will have the smaller prediction error on size estimation.

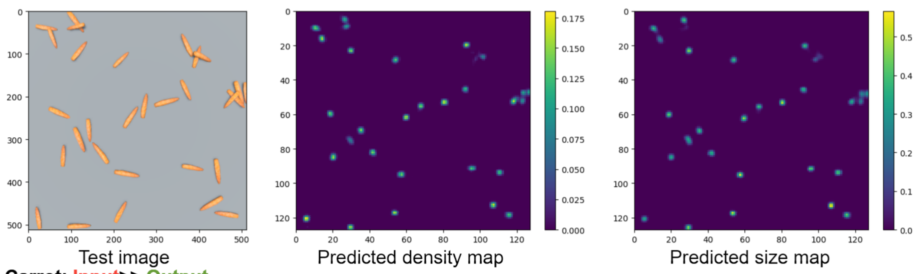
Previously mentioned experiments in this subsection prove that 1) our proposed approach could solve the problems of single-class object counting and size estimation, 2) the created dataset could be used to train and test for such problems.

6.2. Results of multi-class object counting and size estimation

In the experiments of multi-class object counting and size estimation, we create 4 datasets and each dataset contains 3 classes of fruit or vegetable. The results of the 4 experiments are listed in Table 5. From the table, we can see the results from the experiment of 3-class object counting and size estimation close to the mean of each class's results from the experiments of single-class object counting and size estimation in Table 3. For example, the result $MAE_{counting}$ and



Tomato: Input>> Output
Single-class object counting problem:
How many tomatos are there in the test image? (29.32)
Individual object size retrieval reults:
What is the position and size (x axis, y axis, size) of each tomato in the test image?
([(7.0, 2.0, 2.51), (27.0, 4.0, 3.10), (27.0, 10.0, 2.44), (13.0, 15.0, 2.51), (92.0, 21.0, 2.90), (28.0, 22.0, 3.07), (97.0, 27.0, 2.58), (103.0, 27.0, 2.55), (54.0, 28.0, 2.33), (90.0, 44.0, 2.55), (120.0, 47.0, 3.15), (126.0, 48.0, 2.25), (113.0, 49.0, 2.54), (78.0, 52.0, 3.12), (122.0, 53.0, 2.83), (66.0, 57.0, 2.83), (19.0, 61.0, 3.02), (58.0, 62.0, 2.71), (35.0, 69.0, 2.85), (42.0, 82.0, 2.94), (18.0, 84.0, 2.48), (97.0, 90.0, 2.74), (112.0, 93.0, 2.66), (56.0, 95.0, 2.96), (107.0, 113.0, 3.15), (115.0, 119.0, 2.71), (2.0, 120.0, 2.66), (50.0, 124.0, 2.68)])



Carrot: Input>> Output
Single-class object counting problem:
How many carrots are there in the test image? (28.27)
Individual object size retrieval reults:
What is the position and size (x axis, y axis, size) of each carrot in the test image?
([(127.0, 43.0, 0.05), (127.0, 48.0, 2.36), (27.0, 5.0, 2.79), (28.0, 9.0, 0.41), (10.0, 10.0, 2.81), (14.0, 16.0, 2.07), (92.0, 20.0, 2.84), (29.0, 23.0, 3.18), (100.0, 26.0, 1.20), (54.0, 28.0, 2.58), (98.0, 28.0, 1.16), (91.0, 45.0, 2.52), (123.0, 48.0, 2.69), (118.0, 52.0, 2.51), (123.0, 52.0, 1.95), (80.0, 53.0, 2.97), (67.0, 55.0, 2.59), (18.0, 60.0, 2.85), (59.0, 62.0, 3.29), (35.0, 69.0, 3.00), (29.0, 74.0, 3.22), (20.0, 85.0, 2.30), (95.0, 91.0, 2.66), (110.0, 94.0, 2.90), (57.0, 95.0, 3.19), (106.0, 113.0, 3.72), (53.0, 117.0, 3.12), (115.0, 118.0, 3.00), (5.0, 120.0, 2.11), (29.0, 125.0, 2.95)])

Figure 6: The example output results of tomato (best performance) and carrot (worst performance) for one of their test images respectively.

Table 5: Results of multi-class counting and size estimation.

Class	Counting ↓			Retrieval ↑			Size estimation ↓
	MAE_{count}	$RMSE_{count}$	$MAPE_{count}(\%)$	$Precision(\%)$	$Recall(\%)$	$F1(\%)$	$MAPE_{size}(\%)$
Banana, orange & apple	1.21 ± 0.12	1.00 ± 0.18	5.23 ± 0.36	67.25 ± 6.59	98.90 ± 0.25	79.54 ± 4.52	13.34 ± 0.11
Banana, avocado & apple	1.56 ± 0.19	1.18 ± 0.15	6.66 ± 0.82	70.21 ± 3.95	98.65 ± 0.50	81.37 ± 2.51	13.91 ± 0.40
Carrot, pear & melon	1.84 ± 0.34	1.69 ± 0.34	7.73 ± 1.38	67.76 ± 1.26	97.69 ± 0.11	79.41 ± 0.71	16.67 ± 1.06
Banana, avocado & artichoke	1.93 ± 0.11	1.41 ± 0.08	8.28 ± 0.39	64.14 ± 7.11	98.11 ± 0.17	76.90 ± 5.21	15.86 ± 1.46

Table 6: Results of single-class (banana) counting and size estimation on the different number of training images.

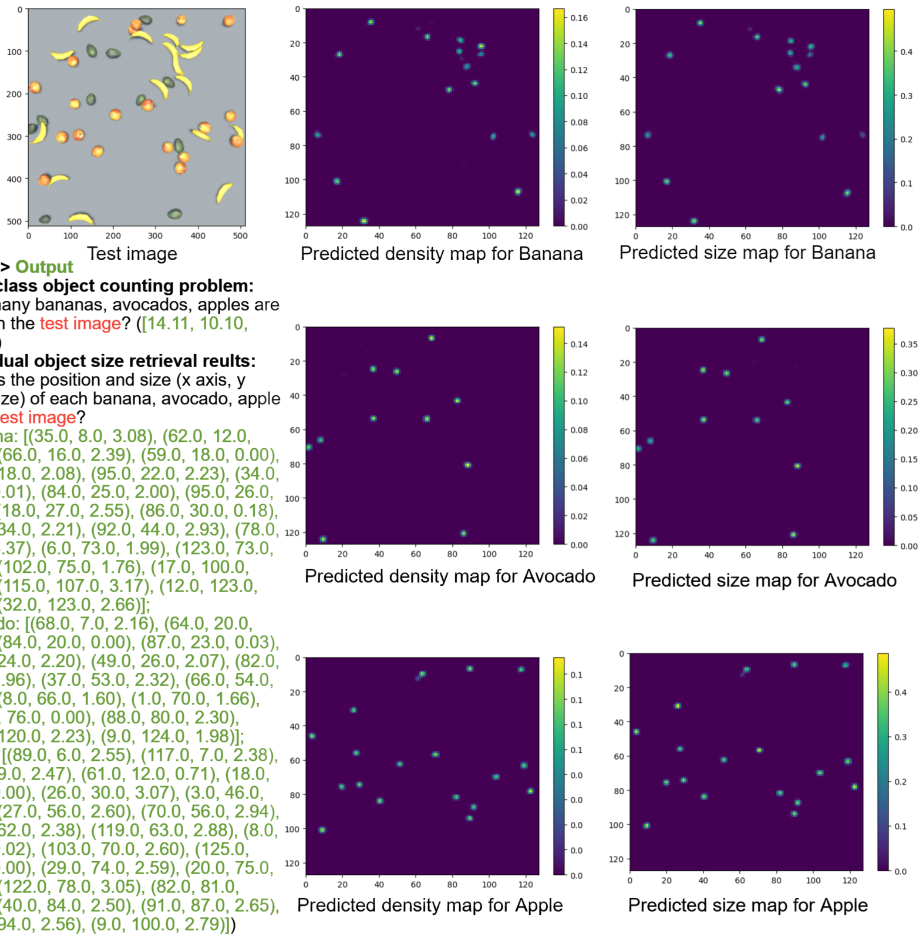
Pictures number		Counting ↓			Retrieval ↑			Size estimation ↓
Train	Test	MAE_{count}	$RMSE_{count}$	$MAPE_{count}(\%)$	$Precision(\%)$	$Recall(\%)$	$F1(\%)$	$MAPE_{size}(\%)$
6	6	15.29 ± 8.96	18.20 ± 10.31	23.09 ± 13.58	68.16 ± 3.76	94.82 ± 1.15	77.84 ± 2.97	36.33 ± 3.50
18	12	3.41 ± 0.48	4.55 ± 0.63	5.11 ± 0.93	86.36 ± 1.72	95.43 ± 1.14	90.18 ± 0.73	20.51 ± 3.64
30	18	3.15 ± 0.22	3.94 ± 0.32	4.35 ± 0.23	86.15 ± 0.84	96.78 ± 0.53	90.96 ± 0.52	12.04 ± 0.20
42	30	1.69 ± 0.14	2.13 ± 0.18	2.41 ± 0.22	85.09 ± 1.97	96.97 ± 0.06	90.35 ± 1.12	10.70 ± 0.63
60	30	1.91 ± 0.33	2.41 ± 0.36	2.60 ± 0.38	85.94 ± 2.40	97.32 ± 0.28	91.00 ± 1.45	9.72 ± 0.31

$MAPE_{size}$ from the experiment "*banana*, *orange* & *apple*" in Table 5 are better than $MAE_{counting}$ and $MAPE_{size}$ from the experiment "*banana*" but worse than the experiment "*orange*" in Table 3. By replacing the *avocado* with *orange* in the dataset (*orange* has better performance in the single-class experiment), both $MAE_{counting}$ and $MAPE_{size}$ are increased. By replacing the *apple* with *artichoke*, both $MAE_{counting}$ and $MAPE_{size}$ are increased even artichoke has better performance in the single-class experiment than the apple. This may be because of the colour effect that both the *avocado* and *artichoke* are green which will affect the prediction result. Figure 7 shows the example output results of experiment of *banana*, *avocado* & *apple* for one of the test images.

All these results prove our proposed deep model architecture-based approach can solve the problems of multi-class object counting and size estimation.

6.3. On the number of training images

To investigate the effect of the number of training images on the prediction performance of our approach, we generate 5 image datasets that have different numbers of training images. The results of the 5 experiments are shown in Table 6. We can see that in general with the increase in the number of training images used in the experiment, both $MAE_{counting}$ and $MAPE_{size}$ are decreased which means the prediction performance of our approach becomes better with the sufficient training dataset. However, keep including more training images does not necessarily



Input>> Output

Multi-class object counting problem:
How many bananas, avocados, apples are there in the test image? ([14.11, 10.10, 17.23])

Individual object size retrieval reults:
What is the position and size (x axis, y axis, size) of each banana, avocado, apple in the test image?
(Banana: [(35.0, 8.0, 3.08), (62.0, 12.0, 0.29), (66.0, 16.0, 2.39), (59.0, 18.0, 0.00), (84.0, 18.0, 2.08), (95.0, 22.0, 2.23), (34.0, 25.0, 0.01), (84.0, 25.0, 2.00), (95.0, 26.0, 1.11), (18.0, 27.0, 2.55), (86.0, 30.0, 0.18), (88.0, 34.0, 2.21), (92.0, 44.0, 2.93), (78.0, 47.0, 3.37), (6.0, 73.0, 1.99), (123.0, 73.0, 0.90), (102.0, 75.0, 1.76), (17.0, 100.0, 2.63), (115.0, 107.0, 3.17), (12.0, 123.0, 0.00), (32.0, 123.0, 2.66)];
Avocado: [(68.0, 7.0, 2.16), (64.0, 20.0, 0.04), (84.0, 20.0, 0.00), (87.0, 23.0, 0.03), (37.0, 24.0, 2.20), (49.0, 26.0, 2.07), (82.0, 43.0, 1.96), (37.0, 53.0, 2.32), (66.0, 54.0, 1.91), (8.0, 66.0, 1.60), (1.0, 70.0, 1.66), (123.0, 76.0, 0.00), (88.0, 80.0, 2.30), (86.0, 120.0, 2.23), (9.0, 124.0, 1.98)];
Apple: [(89.0, 6.0, 2.55), (117.0, 7.0, 2.38), (63.0, 9.0, 2.47), (61.0, 12.0, 0.71), (18.0, 29.0, 0.00), (26.0, 30.0, 3.07), (3.0, 46.0, 2.87), (27.0, 56.0, 2.60), (70.0, 56.0, 2.94), (51.0, 62.0, 2.38), (119.0, 63.0, 2.88), (8.0, 68.0, 0.02), (103.0, 70.0, 2.60), (125.0, 72.0, 0.00), (29.0, 74.0, 2.59), (20.0, 75.0, 2.52), (122.0, 78.0, 3.05), (82.0, 81.0, 2.56), (40.0, 84.0, 2.50), (91.0, 87.0, 2.65), (89.0, 94.0, 2.56), (9.0, 100.0, 2.79)])

Figure 7: The example output results of experiment of *banana*, *avocado* & *apple* for one of the test images.

Table 7: Results of single-class (carrot) counting and size estimation on different backbone networks.

Method	Counting ↓			Retrieval ↑			Size estimation ↓
	MAE_{count}	$RMSE_{count}$	$MAPE_{count}(\%)$	$Precision(\%)$	$Recall(\%)$	$F1(\%)$	$MAPE_{size}(\%)$
TEDNet	1.31 ± 0.18	1.61 ± 0.16	2.06 ± 0.24	93.63 ± 1.49	97.38 ± 0.11	95.35 ± 0.77	11.09 ± 1.45
Inception-v3	4.42 ± 1.01	5.41 ± 1.32	7.34 ± 2.07	88.24 ± 4.33	92.66 ± 0.64	90.19 ± 2.12	13.93 ± 1.63
VGG-16-bn	5.04 ± 0.71	6.11 ± 0.81	7.43 ± 0.88	85.14 ± 2.22	94.95 ± 0.44	89.28 ± 1.34	15.49 ± 1.30
Resnet-50	8.51 ± 2.24	10.71 ± 2.90	11.99 ± 2.80	N.A	N.A	N.A	N.A
Densenet-121	12.72 ± 3.63	15.67 ± 4.85	18.26 ± 3.18	N.A	N.A	N.A	N.A
ShuffleNet v2x0.5	22.86 ± 1.51	25.40 ± 1.79	32.72 ± 2.02	N.A	N.A	N.A	N.A

increase the performance of the prediction. This may be because the model quickly converges close to its performance limit with the increasing number of training images. As shown in Table 6, $MAE_{counting}$ is increased from 1.69 to 1.91 even when we add 18 additional images to the training dataset. For the counting problem, the prediction performance is acceptable even after we only adopt a few images for training: $MAE_{counting}$ of 15.29 for 6 training images and $MAE_{counting}$ of 3.41 for 18 training images. In comparison, there need at least 18 images for training to obtain a high $Precision$ rate of 86.36 and $MAPE_{size}$ of 20.51. This is because the size is an added feature for the size estimation problem where the counting problem does not consider.

The results of this experiment show that our proposed approach and dataset follow the general principle of the deep model: an increase in the training data could improve the prediction performance. This proves the feasibility of our proposed approach and the dataset.

6.4. On the backbone networks

Table 7 shows the experimental results of different backbone CNN models in our approach. **TEDNet** and **Inception-v3** achieve the best and second performance in both the counting and size estimation problems respectively. The training time of **TEDNet** is around 21 minutes. **Inception-v3** and **VGG-16-bn** cost around 26 and 28 minutes for training respectively. **Resnet-50** is faster which only costs around 20 minutes for training but the counting results is not good enough compared with **Inception-v3** and **VGG-16-bn**. We know that the **TEDNet** employs the $Inception-style$ modules. The good performance of **TEDNet** and **Inception-v3** shows that the inception modules are beneficial to crowd counting. The reason is that the inception module was designed to capture different scales of contextual information in each convolutional layer. From the table, we can also observe that the size map predicted by **Resnet -50**, **Densenet-121**, and **ShuffleNet v2x0.5** cannot apply Algorithm 1 for the individual object size retrieval due to the following reasons. On the one hand, that is because the weight of the patched kernel in their obtained size map is not Gaussian distributed but uniformly distributed. For the counting problem, the output result is straightforward to get - the sum of the density map. However, for the size estimation problem, retrieval of the individual objects from the size map predicted by these models is difficult, especially without the ground truth information. On the other hand, the output size of the predicted size map from **Resnet-50** and **Densenet-121** is 64*64 which is too small compared with the patch size (4*4) to do the segmentation. The shrink ratio caused too many touching and covering objects which is hard or impossible to do the segmentation. The performance of **ShuffleNet v2x0.5** is poor. Hence these three models are not suitable for the size estimation problem in our approach. Therefore the size estimation results of these three models in Table 7 are shown as "NA" not available. This experiment evaluates the feasibility and efficiency of the state-of-art backbone CNNs in our approach, which proves our claim of solving multi-class object counting and size estimation problems.

To prove the precision and feasibility of our proposed Algorithm 1 (IOSR) for the individual object detection problem in our approach, we implement the MCNN, VGG-16-bn, and TEDNet with the *apple* and *carrot* datasets. The reasons to select MCNN as comparison are the following: it can output the appropriate size of the size map; the obtained size map cannot be applied with Algorithm 1 because its size map also reflects the shape of objects from the original map. Figure 8 shows an example of the size map obtained from MCNN. We could see that the objects touching and covering are transferred from the original image to the predicted size map. CCL and Watershed algorithms may solve this problem to segment the individual objects. As said before, they do not have the capability of matching the segmented individual objects with the ground truth, the Algorithm 2 (TM) is also applied to identify the TP, FP, and FN instances from the list of segmented individual objects.

From Table 8, we can draw the following conclusions. The method which shows the best performance in each experiment is highlighted in **bold**. Notice that, the *Recall* used to evaluate the result of our method is the ratio of true positive of detected objects and the ground truth, see Eq (11) for detail. By contrast, the *Object detection rate* used to evaluate the result of CCL and Watershed algorithms is the ratio of all detected objects and the ground truth. Comparing with CCL and Watershed algorithm, our method **IOSR+TM** returns the best result for both the *apple* and *carrot* datasets in all experiments. We suppose the reason for these results is

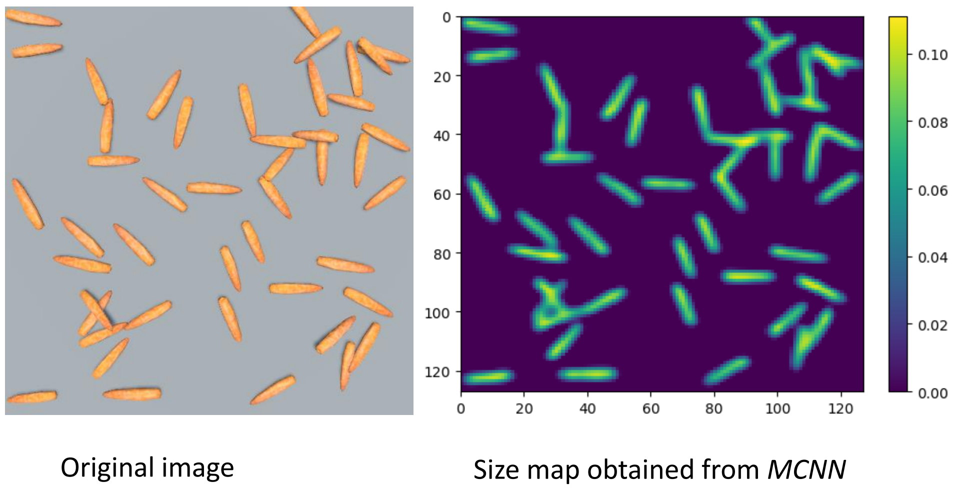


Figure 8: The example of a size map obtained by *MCNN*.

that CCL and Watershed algorithms are traditional segment algorithms that can not better deal with covering and touching items. The covering or touching objects will be segmented as one object. CCL and Watershed algorithms perform better for round objects. The *Object detection rates* calculated by these two methods for the dataset *apple* are much better than *carrot* under all models. For example, using the CCL method, we obtain 90.82% for the *apple* with the model MCNN and 65.71 % for the *carrot* with the same model. This shows the limitation of the CCL and Watershed algorithm and proves our assumption. The results of **IOSR+TM** do not show this drawback. The result *Recall* of **IOSR+TM** has the better performance with the model **TEDNet** compared with **VGG-16-bn**. This might be because the output size of **TEDNet** is 512*512 which could avoid more touching and covering patches. As shown in Table 7, **TEDNet** shows the better performance for the size estimation. The model that outputs a size map like MCNN is not appropriate to do the size estimation in our approach since the object touching and covering are transferred from the original image to the predicted size map. CCL and Watershed algorithms also do not show a good performance on segmenting such kind of size map.

In conclusion, the results from Table 8 show our proposed individual object size retrieval method can output a better performance to identify the individual object and calculate its size from the predicted size map of a text image compare with existing methods. This proves our

Table 8: Results of the different object detecting methods for size map obtained by different backbone CNN model.

Dataset	Model	Object detecting method	Object detection rate (D,%) / Recall (R,%) ↑	Output size
Apple	MCNN	Watershed+TM	87.06 ± 1.86 (D)	128*128
Apple	MCNN	CCL+TM	90.82 ± 1.23 (D)	128*128
Apple	VGG-16-bn	CCL+TM	89.23 ± 2.16 (D)	128*128
Apple	VGG-16-bn	Watershed+TM	82.57 ± 5.07 (D)	128*128
Apple	VGG-16-bn	**IOSR+TM**	**98.72 ± 0.35 (R)**	128*128
Apple	TEDNet	CCL+TM	99.06 ± 1.16 (D)	512*512
Apple	TEDNet	Watershed+TM	97.68 ± 0.19 (D)	512*512
Apple	TEDNet	**IOSR+TM**	**99.75 ± 0.05 (R)**	512*512
Carrot	MCNN	Watershed+TM	64.00 ± 6.51 (D)	128*128
Carrot	MCNN	CC+TML	65.71 ± 15.94 (D)	128*128
Carrot	VGG-16-bn	CCL+TM	86.00 ± 2.11 (D)	128*128
Carrot	VGG-16-bn	Watershed+TM	76.34 ± 1.87 (D)	128*128
Carrot	VGG-16-bn	**IOSR+TM**	**94.95 ± 0.44 (R)**	128*128
Carrot	TEDNet	CCL+TM	86.02 ± 1.58 (D)	512*512
Carrot	TEDNet	Watershed+TM	85.91 ± 1.93 (D)	512*512
Carrot	TEDNet	**IOSR+TM**	**97.38 ± 0.11 (R)**	512*512

claim of overcoming the barrier in current research of multi-class object size estimation in Section 1.

7. Conclusion

In this work, we formulate a practical problem of multi-class object counting and size estimation. A unified framework based on deep convolutional neural networks is proposed. We create a new dataset for evaluating the performance in multi-class object counting and size estimation by synthetically generating images of multi-class objects using the Unity platform. Experimental results demonstrate the effectiveness of the proposed solution and provide baseline performance for the new problem. It is also discovered different deep CNN models are suitable for counting objects of different shapes and appearances. In addition, the dataset could provide a suitable testbed for few-shot learning and meta-learning beyond image classification tasks.

Our work can be extended in several aspects. First, our dataset is synthetically generated from the Unity platform - a gaming engine. We have discussed our dataset could be used to train the deep learning model and this model can be used in the real applications of object counting and size estimation without further feature-text-specified training. Some work also proves the use of data from the gaming environment can be transferred to real applications or other gaming environments. Due to the lack of a dataset from the real world, we can not carry out such an experiment to test our dataset and the proposed approach. To overcome this limitation, we plan to collaborate with a supermarket company to get some real-world data, i.e., collect the images of fruit and vegetable brought by the customer from the self-checkout machine. Second, different background colours and light can be the environmental parameters that affect the performance of our proposed approach. The further experiment needs to be carried out for testing these environmental effects for the training and test performance. In Unity, the background can be changed to any colour or image by replacing the material of the panel. Also, a well-designed lighting model could simulate realistic light. Following the above, we could generate a new dataset for the environmental effect experiment.

Although we used 13 classes of fruit or vegetable (and also can be extended by creating a new model in Unity) to train the deep learning model, how to deal with the new images which contain the new class of the object but do not provide the marking information? To deal with such situations, in the future, we will focus on few-shot learning based on the created dataset. How to improve the prediction performance under the limited number of training images is a research problem worth further investigation. We plan to explore meta-learning in this direction of research. Current few-shot learning and meta-learning methods mostly focus on image classification tasks. Instead, object counting and size estimation in our approach are regression problems. Therefore, how to use meta-learning for the regression tasks under the input of the images is another research direction in the future beyond this work.

References

[1] Aich, S. and Stavness, I. (2017). Leaf counting with deep convolutional and deconvolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2080–2089.

[2] Al-Thani, N., Albuainain, A., Alnaimi, F., and Zorba, N. (2020). Drones for sheep livestock monitoring. In *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, pages 672–676. IEEE.

[3] Apolo-Apolo, O., Martínez-Guanter, J., Egea, G., Raja, P., and Pérez-Ruiz, M. (2020). Deep learning techniques for estimation of the yield and size of citrus fruits using a uav. *European Journal of Agronomy*, 115:126030.

[4] Cang, Y., He, H., and Qiao, Y. (2019). An intelligent pig weights estimate method based on deep learning in sow stall environments. *IEEE Access*, 7:164867–164875.

[5] Cao, L., Xiao, Z., Liao, X., Yao, Y., Wu, K., Mu, J., Li, J., and Pu, H. (2021). Automated chicken counting in surveillance camera environments based on the point supervision algorithm: Lc-densefcn. *Agriculture*, 11(6):493.

[6] Cook, D. J., Holder, L. B., and Youngblood, G. M. (2007). Graph-based analysis of human transfer learning using a game testbed. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1465–1478.

[7] Ege, T., Ando, Y., Tanno, R., Shimoda, W., and Yanai, K. (2019). Image-based estimation of real food size for accurate food calorie estimation. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 274–279. IEEE.

[8] Enozone (2018). Supermarket with lod. https://assetstore.unity.com/packages/3d/environments/urban/supermarket-with-lod-110050. Accessed: 2020-09-30.

[9] Gao, G., Liu, Q., and Wang, Y. (2020). Counting from sky: A large-scale data set for remote sensing object counting and a benchmark method. *IEEE Transactions on Geoscience and Remote Sensing*, 59(5):3642–3655.

[10] Garcia Arnal Barbedo, J. (2012). A review on methods for automatic counting of objects in digital images. *IEEE Latin America Transactions*, 10(5):2112–2124.

[11] Gené-Mola, J., Sanz-Cortiella, R., Rosell-Polo, J. R., Escolà, A., and Gregorio, E. (2021). In-field apple size estimation using photogrammetry-derived 3d point clouds: Comparison of 4 different methods considering fruit occlusions. *Computers and Electronics in Agriculture*, 188:106343.

[12] Go, H., Byun, J., Park, B., Choi, M.-A., Yoo, S., and Kim, C. (2021). Fine-grained multi-class object counting. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 509–513. IEEE.

[13] Gongal, A., Karkee, M., and Amatya, S. (2018). Apple fruit size estimation using a 3d machine vision system. *Information Processing in Agriculture*, 5(4):498–503.

[14] Guo, D., Li, K., Zha, Z.-J., and Wang, M. (2019). Dadnet: Dilated-attention-deformable convnet for crowd counting. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 1823–1832.

[15] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[16] Heinrich, K., Roth, A., and Zschech, P. (2019). Everything counts: a taxonomy of deep learning approaches for object counting. In *ECIS*.

[17] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[18] Idrees, H., Saleemi, I., Seibert, C., and Shah, M. (2013). Multi-source multi-scale counting in extremely dense crowd images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2547–2554.

[19] Idrees, H., Tayyab, M., Athrey, K., Zhang, D., Al-Maadeed, S., Rajpoot, N., and Shah, M. (2018). Composition loss for counting, density map estimation and localization in dense crowds. In *Proceedings of the European conference on computer vision (ECCV)*, pages 532–546.

[20] Jiang, X., Xiao, Z., Zhang, B., Zhen, X., Cao, X., Doermann, D., and Shao, L. (2019). Crowd counting and density estimation by trellis encoder-decoder networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6133–6142.

[21] Jingying, W. (2021). A survey on crowd counting methods and datasets. In *Advances in Computer, Communication and Computational Sciences*, pages 851–863. Springer.

[22] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and

Lange, D. (2020). Unity: A general platform for intelligent agents.

[23] Lempitsky, V. and Zisserman, A. (2010). Learning to count objects in images. *Advances in neural information processing systems*, 23.

[24] Liu, X., Chen, S. W., Aditya, S., Sivakumar, N., Dcunha, S., Qu, C., Taylor, C. J., Das, J., and Kumar, V. (2018). Robust fruit counting: Combining deep learning, tracking, and structure from motion. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1045–1052. IEEE.

[25] Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131.

[26] Ma, T., Liu, S., and Wang, Q. (2022). On data annotation efficiency for image based crowd counting. In *International Conference on Visual Communications and Image Processing (VCIP)*. IEEE.

[27] Najman, L. and Schmitt, M. (1994). Watershed of a continuous function. *Signal Processing*, 38(1):99–112.

[28] Okamoto, K. and Yanai, K. (2016). An automatic calorie estimation system of food images on a smartphone. In *Proceedings of the 2nd International Workshop on Multimedia Assisted Dietary Management*, pages 63–70.

[29] Oo, L. M. and Aung, N. Z. (2018). A simple and efficient method for automatic strawberry shape and size estimation and classification. *Biosystems engineering*, 170:96–107.

[30] Pandit, A. and Rangole, J. (2014). Literature review on object counting using image processing techniques. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 3(4):8509–8512.

[31] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *Thirty-first Conference on Neural Information Processing Systems*.

[32] Ponce, J. M., Aquino, A., Millán, B., and Andújar, J. M. (2018). Olive-fruit mass and size estimation using image analysis and feature modeling. *Sensors*, 18(9):2930.

[33] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99.

[34] Sa, J., Choi, Y., Lee, H., Chung, Y., Park, D., and Cho, J. (2019). Fast pig detection with a top-view camera under various illumination conditions. *Symmetry*, 11(2):266.

[35] Salamí, E., Gallardo, A., Skorobogatov, G., and Barrado, C. (2019). On-the-fly olive tree counting using a uas and cloud services. *Remote Sensing*, 11(3):316.

[36] Samet, H. and Tamminen, M. (1988). Efficient component labeling of images of arbitrary dimension represented by linear bintrees. *IEEE transactions on pattern analysis and machine intelligence*, 10(4):579–586.

[37] Shaker, N. and Abou-Zleikha, M. (2016). Transfer learning for cross-game prediction of player experience. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

[38] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[39] Sindagi, V. A. and Patel, V. M. (2018). A survey of recent advances in cnn-based single image crowd counting and density estimation. *Pattern Recognition Letters*, 107:3–16. Video Surveillance-oriented Biometrics.

[40] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826.

[41] Tong, P., Han, P., Li, S., Li, N., Bu, S., Li, Q., and Li, K. (2021). Counting trees with point-wise supervised

segmentation network. *Engineering Applications of Artificial Intelligence*, 100:104172.

[42] Wang, Q. and Breckon, T. P. (2022). Crowd counting via segmentation guided attention networks and curriculum loss. *IEEE Transactions on Intelligent Transportation Systems*.

[43] Wang, Z., Walsh, K. B., and Verma, B. (2017). On-tree mango fruit size estimation using rgb-d images. *Sensors*, 17(12):2738.

[44] Wei, Y., Tran, S., Xu, S., Kang, B., and Springer, M. (2020). Deep learning for retail product recognition: Challenges and techniques. *Computational intelligence and neuroscience*, 2020.

[45] Xia, G.-S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M., and Zhang, L. (2018). Dota: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3974–3983.

[46] Xu, W., Liang, D., Zheng, Y., Xie, J., and Ma, Z. (2021). Dilated-scale-aware category-attention convnet for multi-class object counting. *IEEE Signal Processing Letters*.

[47] Yang, Y., Li, G., Wu, Z., Su, L., Huang, Q., and Sebe, N. (2020). Reverse perspective network for perspective-aware object counting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4374–4383.

[48] Zhang, A., Shen, J., Xiao, Z., Zhu, F., Zhen, X., Cao, X., and Shao, L. (2019). Relational attention network for crowd counting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6788–6797.

[49] Zhang, B., Wang, N., Zhao, Z., Abraham, A., and Liu, H. (2021). Crowd counting based on attention-guided multi-scale fusion networks. *Neurocomputing*, 451:12–24.

[50] Zhang, G., Pan, Y., Zhang, L., and Tiong, R. L. K. (2020). Cross-scale generative adversarial network for crowd density estimation from images. *Engineering Applications of Artificial Intelligence*, 94:103777.

[51] Zhang, Y., Zhou, D., Chen, S., Gao, S., and Ma, Y. (2016). Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 589–597.

[52] Zhu, P., Wen, L., Du, D., Bian, X., Ling, H., Hu, Q., Wu, H., Nie, Q., Cheng, H., Liu, C., et al. (2018). Visdrone-vdt2018: The vision meets drone video detection and tracking challenge results. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0.