A Web Application for

Fruit Identifying and Counting

Based on Deep Learning

COMP90055 Computing Project (25 Credits)
Type of Project: Software Development Project

Supervisor: Prof. Richard Sinnott

Huaqing Yu	930285
Shining Song	887007
Shaoxi Ma	888378

Semester 1, 2019

Abstract

Deep learning is one of the most important breakthroughs in artificial intelligence in the past decade. It brings significant improvements to technologies such as computer vision, image recognition, and object detection.

This project aims to achieve object detection of images containing fruits by means of convolutional neural networks. We built a dataset containing images of different varieties of fruit. Then, based on this dataset, we applied Faster R-CNN and SSD technologies to build the models separately. By comparing the two models and analysing the experimental results, we selected Faster R-CNN that is more suitable for this project to build our final model. This model is effective for detecting a variety of fruits such as lime and apple.

Based on the Faster R-CNN model applied in this project, we developed a web application that allows users to take or upload an image, then identify and label the possible fruits in the image in seconds and return the total number of fruits.

Keywords: Image Recognition, Object Detection, Deep Learning, Neural Network, TensorFlow, Faster R-CNN, SSD, Web App

We certify that:

- this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university, and that to the best of our knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

- where necessary we have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department.

- the thesis is 6489 words in length (excluding text in images, tables, bibliographies, and appendices).

Acknowledgement

We would like to thank our supervisor Richard Sinnott for providing us such a rare opportunity to enhance our technical and academic skills. In the past three months, Richard has provided irreplaceable assistance for our projects. When we encounter problems and challenges, the email we sent always got a reply from Richard immediately. Richard is also willing to arrange a time to discuss our project with us and never be perfunctory. Knowing that we were lacking training data, Richard even took and sent photos of lemon trees in his garden to help us expand our dataset. We are truly grateful for his help and guidance.

Table of Contents

List of Figures

1 Introduction

1.1. Project Profile and Value

Farmers need to know the growth of fruit trees to estimate production. The traditional method is to manually count the number of fruits on a fruit tree roughly as a basis for judging the yield. This method not only wastes manpower, but may also cause large deviations when counting fruit-intensive trees like Figure 1. Therefore, automatic fruit counter is a project with potential application value. Although image recognition and target detection techniques are relatively mature since deep learning is rapidly evolving, ensuring acceptable speed and accuracy of fruit counter is still the focus and difficulty. Currently there are few studies on fruit identification and detection, so the lack of relevant data sets is a major challenge for this project.

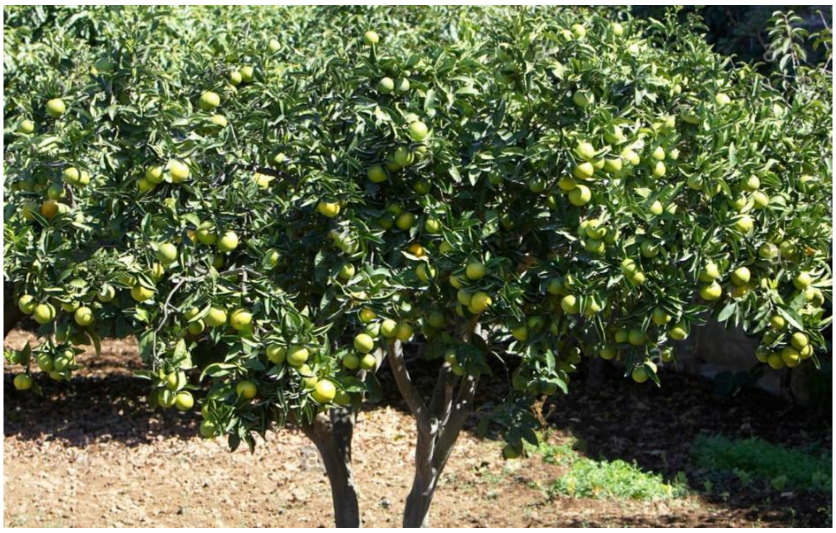


Figure 1 A fruit-intensive tree that is difficult to count manually

1.2. Methodologies

1.2.1. Neural Network

Neural network is an operational model consisting of a large number of nodes (or neurons) connected to each other. Figure 2 shows the process of a simple neural network. Each node represents a specific output function called an activation function. The connection between every two nodes represents a weighting value for passing the connection signal, called weight, which is equivalent to the memory of the neural network. The output of the network varies depending on the connection method of the network, the weight value and the excitation function. The network itself is usually an approximation of an algorithm or function in nature, or it may be an expression of a logic strategy.

With the deepening of the research work of neural networks, many practical problems that are difficult to solve by traditional programming methods have been successfully solved, and good intelligent characteristics have been demonstrated.

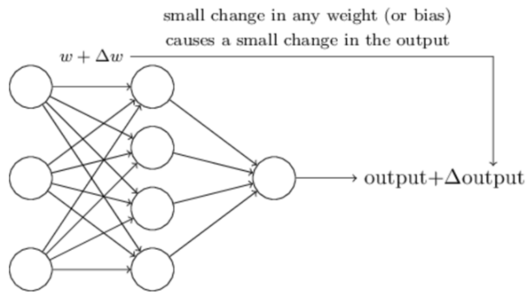

small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

Figure 2 Neural network process

1.2.2. Deep Learning

Deep learning, which applied neural network as its framework, is an algorithm based on the representation and learning of data in machine learning. Different from traditional machine learning, as shown in Figure 3, deep learning extracts complex features from basic features instead of manually feature extraction [1].

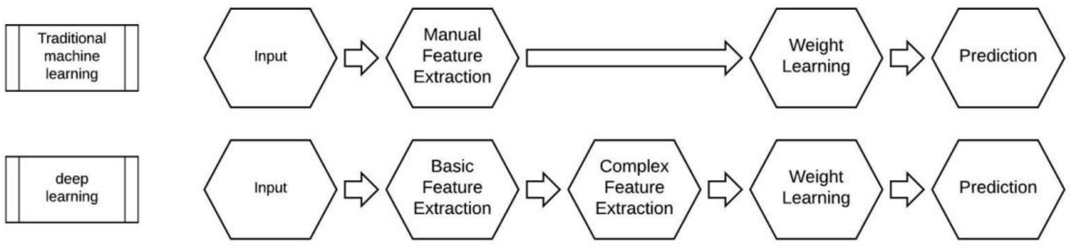


Figure 3 Comparison between machine learning and deep learning

Observations (e.g. images) can be represented in a variety of ways, such as a vector of each pixel intensity value, or more abstractly represented as a series of edges, regions of a particular shape, and the like [2]. Therefore, it is easier to learn tasks from an instance using some specific representation.

1.2.3. Deep Neural Network

A deep neural network is a neural network with at least one hidden layer. Like shallow neural networks, deep neural networks can also provide modelling for complex nonlinear systems, but the extra levels provide a higher level of abstraction for the model, thus improving the model's capabilities [3]. Deep neural networks are usually feed forward neural networks, of which Convolutional Neural Network is one of the most widely used neural networks.

1.2.4. Convolutional Neural Network (CNN)

CNN is a deep learning model which is similar to the multi-layer perceptron of artificial neural network and is a computational network structure that used to deal with local and global correlations. Figure 4 shows the architecture of a standard CNN.

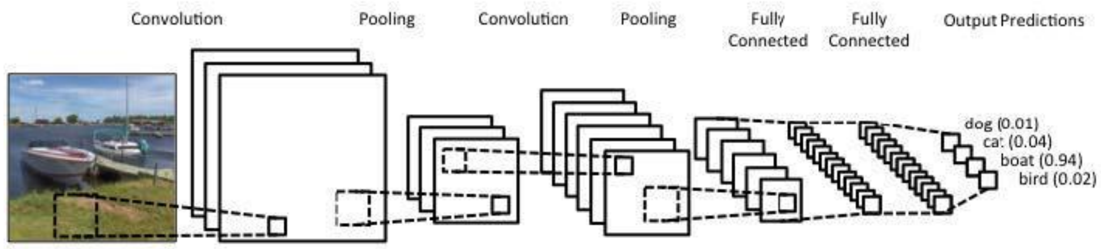


Figure 4 CNN architecture

Since image data has a significant local and global relationship, its application in image recognition and object detection has been extremely successful. In the next section, we will focus on how CNN's derived algorithms are applied to object detection.

1.3. Object Detection

The task of object detection is to find all the objects of interest in the image and determine their category and location. It is one of the core issues in the field of computer vision. Object detection has always been the most challenging problem in computer vision due to the different appearances, shapes and poses of various objects, as well as the interference of illumination and occlusion during imaging [4].

How to parse the information that can be understood by the computer from the image is the central problem of object detection. The deep learning model, due to its powerful representation ability, combined with the accumulation of data and the advancement of computational power, has become a hot research direction for object detection.

For object detection, as shown in Figure 5, there are two classic deep learning models: 2-stage detection model and 1-stage detection model.

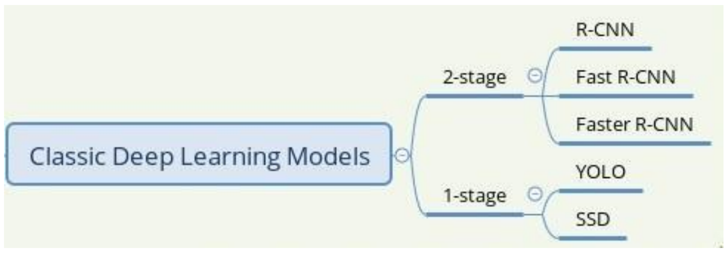


Figure 5 Classification of Object Detection Models

1.3.1. 2-stage Model

The 2-stage model is named after its two-stage processing of images, also known as the Region-based method, and R-CNN and its derivatives are representative of this type.

a) R-CNN

R-CNN abstracts the detection into two processes. The first step is to propose several regions that may contain objects based on the image (i.e. local clipping of the image, called Region Proposal) by Selective Search algorithm. The second step is to run the performing classification network (AlexNet) on the proposed areas to get the categories of objects in each area. The network structure of R-CNN is shown in Figure 6.

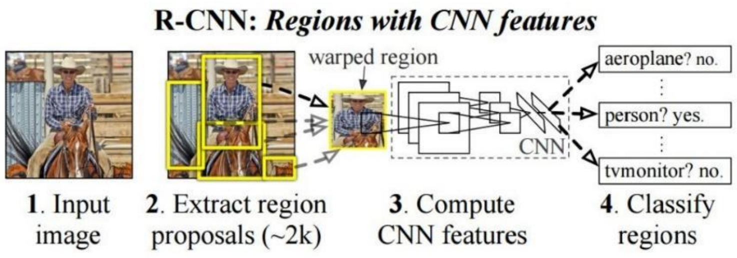


Figure 6 R-CNN network structure

The idea of R-CNN is straightforward. It is the task of transforming the inspection task into a regional classification task. It is the test of the deep learning method on the detection task. There are also many problems in the model itself, such as the need to train three different models (Proposal, Classification, Regression), performance problems caused by excessive double counting, etc.

b) Fast R-CNN

The reason why R-CNN is time consuming is that CNN is performed separately on each Proposal, and there is no shared calculation. In this regard, Fast R-CNN will transfer the basic network to the R-CNN subnet after it has been run on the whole picture, and share most of the calculations. Therefore, it is called Fast R-CNN.

Figure 7 shows the architecture of the Fast R-CNN. The image is obtained by the feature extractor, and the Selective Search algorithm is run on the original image and the Region of Interest (RoI) is mapped to the feature map. Then RoI Pooling is performed for each RoI to obtain the feature vector of the same length. The feature vector performs the collation of positive and negative samples, passes to the parallel R-CNN sub-network, performs classification and regression, and unifies the losses of the two.

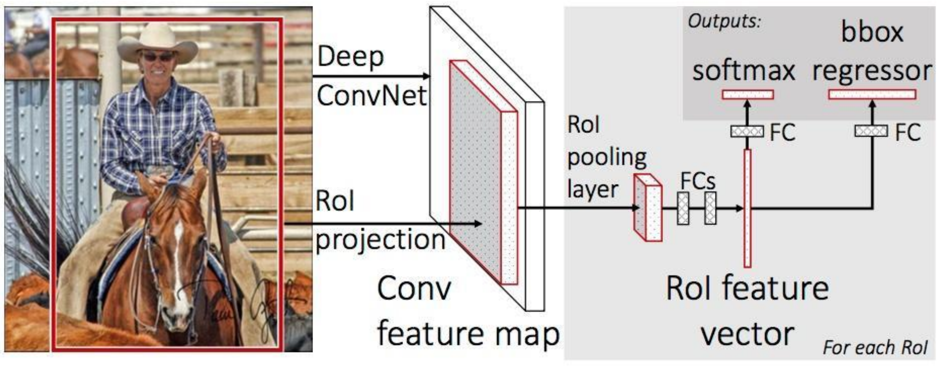


Figure 7 Fast R-CNN network structure

This structure of Fast R-CNN is the prototype of the meta-structure used in the main 2-stage method of the detection task. It unifies Proposal, Feature Extractor, Object Classification and Localization in a whole structure, and improve feature utilization efficiency through shared convolution calculation.

c) Faster R-CNN

Faster R-CNN is the ground-breaking work of the 2-stage method. The greatest achievement is its replacement of the Regional Proposal Networks (RPN) of the Selective Search algorithm, enabling detection tasks to be done end-to-end by the neural network [5]. RPN models the Proposal task as a two-class (whether it is an object) problem.

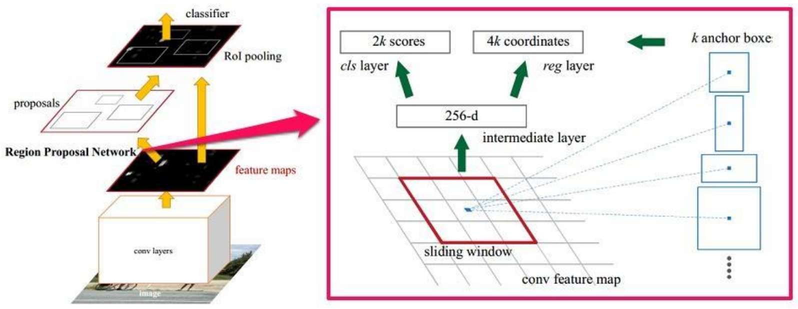


Figure 8 Faster R-CNN network structure

The first step is to generate an anchor box of different size and aspect ratio on a sliding window (as shown in the right part of Figure 8), determine the threshold of IoU, and calibrate the positive and negative of these anchor boxes according to Ground Truth. Thus, the sample data passed into RPN is organized into an anchor box and whether each anchor box has an object (two classification labels). RPN maps each sample to a probability value and four coordinate values [6]. The probability value reflects the probability that the anchor box has an object, and the four coordinate values are used to regress the position of the defined object. Finally, the losses of the two classification and coordinate regression are unified and trained as the target of RPN.

Faster R-CNN is very satisfactory in terms of accuracy and speed. The success of Faster R-CNN is that the "deepening" of the inspection task is done with RPN. The idea of using the sliding window to generate the anchor box is also being used in later work, such as YOLO v2.

1.3.2. 1-stage Model

1-stage model has no intermediate region detection process, and the prediction results are obtained directly from the image, which is also known as the Region-free method.

a) YOLO

You Look Only Once (YOLO) gets its name by processing the image once and getting the position and classification at the same time. It describes the detection task as a unified, end-to-end regression problem [7].

Compared to 2-stage models, YOLO has an obvious speed advantage and its real-time features are impressive. However, YOLO itself has some problems. For example, the mesh is rough, and the number of boxes generated by each mesh limits the detection of small-scale objects and similar objects, which is an unacceptable issue for this project.

b) SSD

The main design of Single Shot Detector (SSD) is to feature layer extraction, and to perform border regression and classification in turn. Because different levels of feature maps can represent different levels of semantic information, low-level feature maps can represent low-level semantic information (with more details), and can also improve the quality of semantic segmentation, suitable for small-scale target learning. High-level feature maps can represent high-level semantic information, smooth segmentation results, and are suitable for in-depth learning of large-scale goals.

The SSD network is divided into 6 stages, each stage can learn a feature map, and then carry out border regression and classification [8]. As shown in Figure 9, the SSD network uses the first five-layer convolutional network of VGG16 as the first stage, and then converts the two fully connected layers in VGG16 into two convolutional layers, Conv6 and Conv7. Then on this basis, SSD network continues to add a four-layer network (Conv8, Conv9, Conv10 and Conv11) to extract higher-level semantic information.

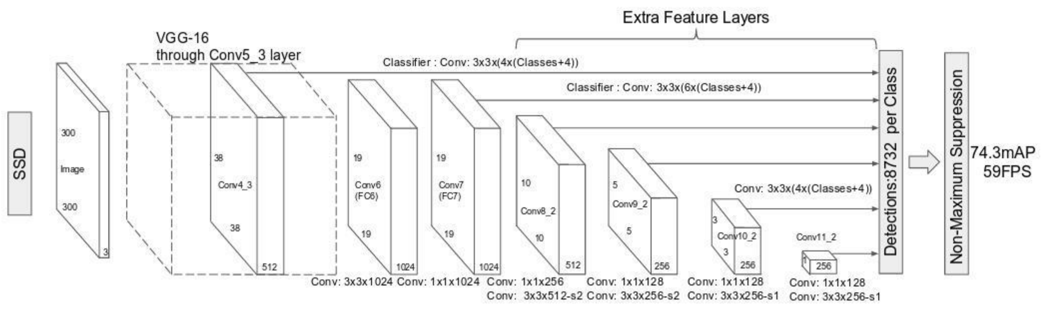


Figure 9 SSD network structure

Compared to YOLO, SSD has multi-scale feature maps. Based on the different convolution segments of VGG, the feature map is output to the regression. This can improve the detection accuracy of small objects, which is of concern in this project.

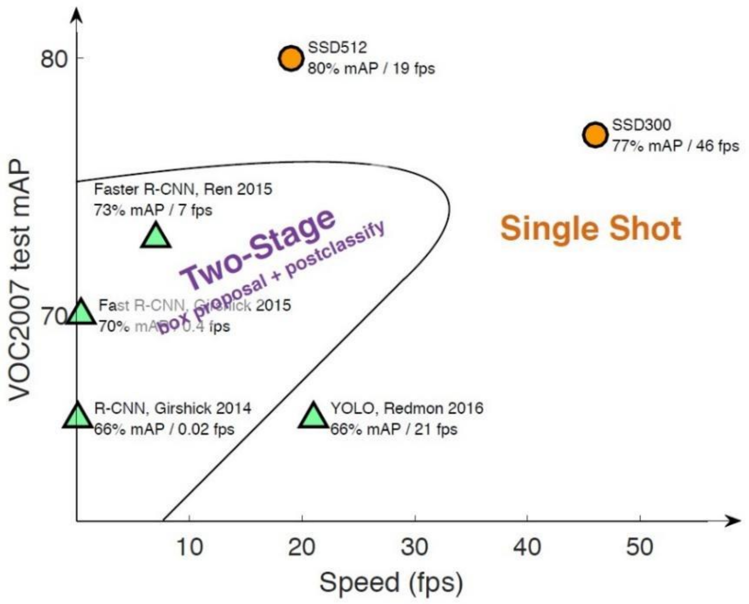


Figure 10 Performance comparison of different detection algorithms

Figure 10 shows the performance comparison of different detection algorithms. Considering both speed and accuracy, we selected Faster R-CNN and SSD to build learning models for this project.

1.4. Web App

After we had finished training our models, we built an application for our client to use in field. To meet the requirement, a client-server model was employed in developing our web application. Since the growers need to estimate the orange production of their growing trees, we implemented a web app including image capturing feature and deployed the object detecting feature on a remote server to process the captured images. This allows users to access our service on any smartphone, e.g. Android or iOS devices, using web browser. The implementation details would be discussed in section 4.

2 Dataset

To build a learning model, we must first build a high-quality data set, which is the basis and key to make the model perform well. The higher the picture quality and the larger the number of pictures, the better the training effect.

Due to limited research and information on fruit identification, we are unable to obtain ready-made data sets. So, we implement the crawler to crawl the image on the web and apply labelImg, which is a Python program that converts images into xml files that can be used by training, to manually label the image for later model training.

2.1. Dataset Source

We crawl 2,000 images from Google Images by google-images-download, which is a python crawler, as the raw data. Since those images contain some “bad” ones (e.g. low resolution, duplicate, watermark, irrelevant, etc.), we need to manually remove those images.

In addition, since the pictures from the network are mostly not close to the photos taken in reality, as a supplement, we take some pictures of limes and oranges to expect the model to better recognize the real photos.

After trimming and supplementing, there are 996 images of variety fruit types in our dataset.

2.2. Dataset Expansion

A well-performing model requires massive image data support. Obviously, the current data set is too small to start training the model. Due to the limited number of high-quality images available from the web and other sources, we need to rely on other means to augment the data set.

2.2.1. Rotation

Rotating images to generate more is the most common way to solve data scarcity issue in object detection. It is worth noting that the angle of the image does not and should not affect the detection of objects. Therefore, we randomly rotate each image 90 degrees or 180 degrees, and Figure 11 shows the rotation of images. When training, the model will treat the two images as completely different images.

In addition, the robustness of the model is improved since more angled samples are added to the data set.

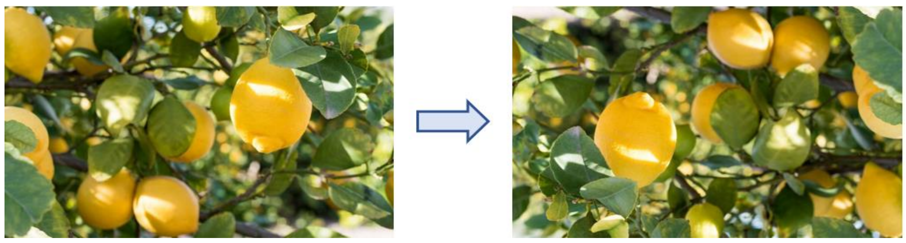


Figure 11 Image rotation

2.2.2. Split

In order to improve the robustness of the model, inputs of different scales are required. Taking SSD as an example, the algorithm actually enhances the data set through the following three training strategies:

a) take the entire image as input;
b) use IOU and the target object of 0.1, 0.3, 0.5, 0.7, and 0.9 as patches. These patches are between the original image size [0.1, 1], and the corresponding aspect ratio is between [0.5, 2];
c) randomly take a patch.

For some large images in our dataset, they may contain dozens or even hundreds of objects. As mentioned above, splitting these images into small images can indirectly expand our data set for better training results.

2.2.3. Colour Adjustment

Different from some algorithms that increase the efficiency by grayscale image processing, both faster R-CNN and SSD we applied in our models are colour-based. Figure 12 shows the comparison between coloured image and uncoloured image on our faster R-CNN model: all 4 apples are detected in coloured image with confidence over 90%, while only 2 apples are detected in uncoloured image with confidence about 50%.

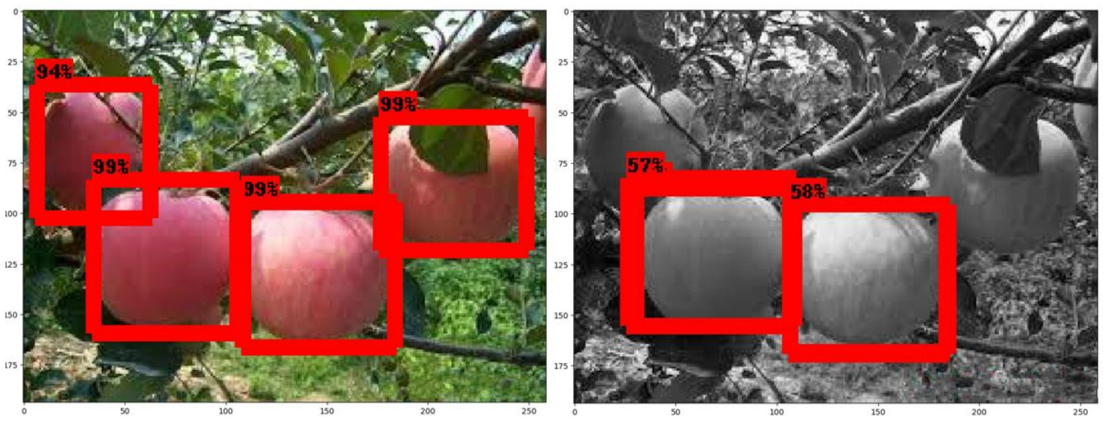


Figure 12 Detection accuracy of coloured image and uncoloured image

Therefore, as shown in Figure 13, we adjust the RGB of the images to generate more images with different brightness and saturation. This can not only expand our dataset, but also improve the performance of the model while detecting images with lower brightness.

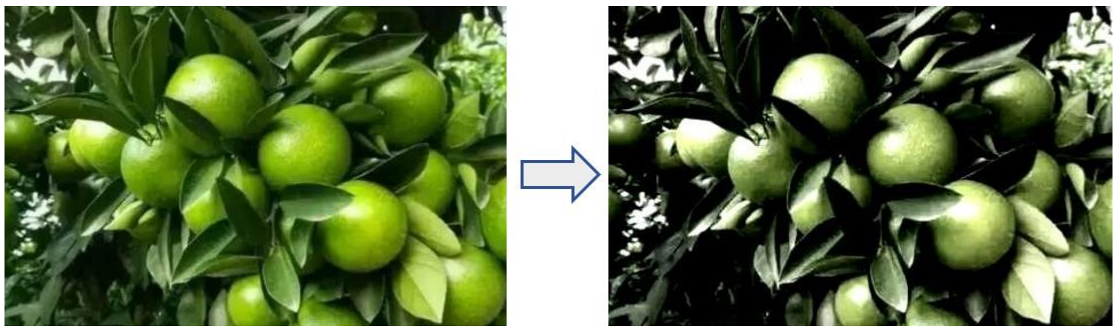
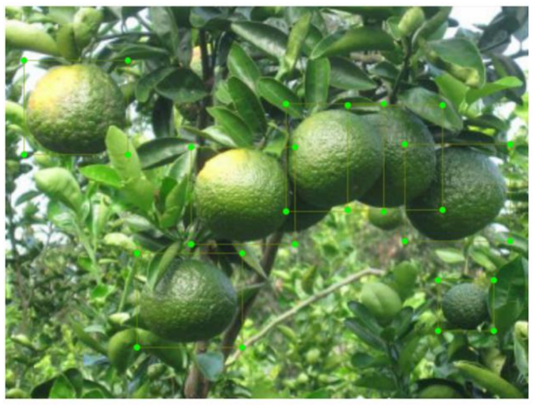
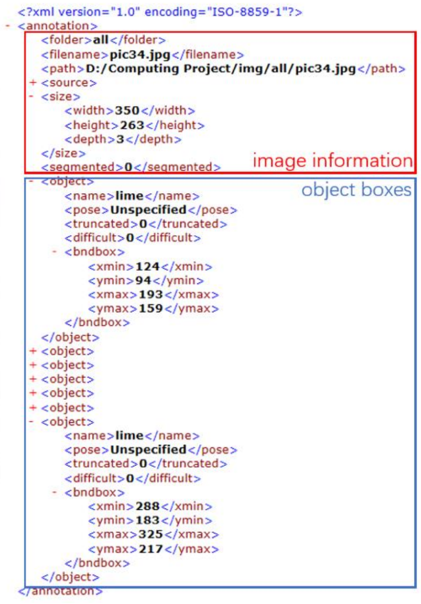


Figure 13 Adjust image RGB by python script

2.3. Dataset Integration

In our dataset, in addition to the images, there are corresponding xml files which are generated by labelImg and contain basic information such as the name and resolution of the image, and a set of coordinates of the boxes that records the position of the label, as shown in Figure 14.

To facilitate subsequent operations, we write a python script to unify the naming rules for images and files.

```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <annotation>
    <folder>all</folder>
    <filename>pic34.jpg</filename>
    <path>D:/Computing Project/img/all/pic34.jpg</path>
  + <source>
  - <size>
      <width>350</width>
      <height>263</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>lime</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>124</xmin>
        <ymin>94</ymin>
        <xmax>193</xmax>
        <ymax>159</ymax>
      </bndbox>
    </object>
  + <object>
  + <object>
  + <object>
  + <object>
  + <object>
  - <object>
      <name>lime</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>288</xmin>
        <ymin>183</ymin>
        <xmax>325</xmax>
        <ymax>217</ymax>
      </bndbox>
    </object>
  </annotation>
```

Figure 14 Image and its corresponding xml file

2.4. Dataset Summary

As shown in Figure 15, the data set we finally used to train the models contains 2,995 images of 6 types of fruit: lime, lemon, apple, mandarin, tomato and orange. In order to test the accuracy of the model, we randomly select 10% of the images (299 images) as the test set.

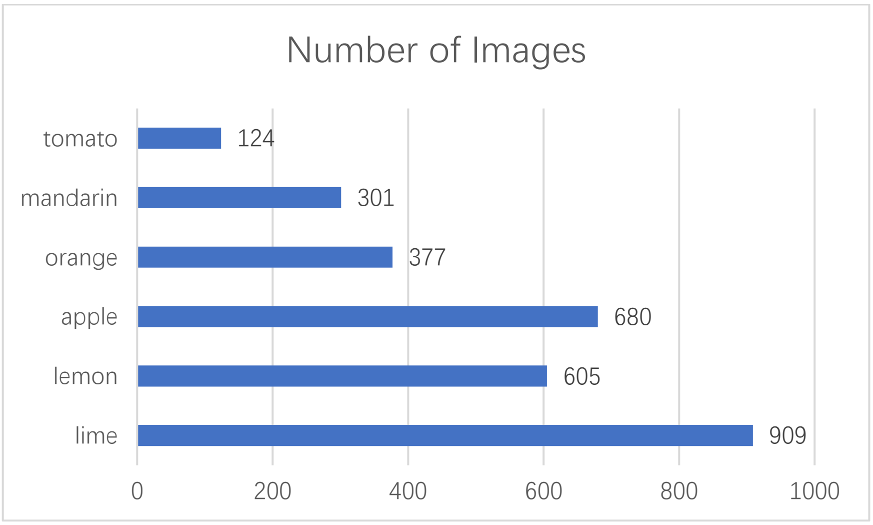


Figure 15 Composition of the dataset

3 Outcome Analysis

3.1. Performance Analysis

After doing some research on the advantages and disadvantages of the common object detection models, ssd_mobilenet_v1 and faster_rcnn_inception_v2 are chosen to be trained in the project with same training samples. The metrics used to evaluate the performance of model are accuracy, speed and loss. To avoid misleading others, we used same hardware platform (i.e. CPU, GPU, RAM), software libraries, and test set to test the learning performance of trained models. The accuracy used in model evaluation is defined as:

$$\left(1 - \frac{Difference}{Manual\ Count}\right) \times 100\%$$

3.1.1. Accuracy Analysis

To stimulate the actual situation, the test images are either taken from gardens or downloaded from Internet which seem like taken by people with high quality, as shown in Figure 16. The detailed results of SSD and Faster R-CNN are recorded in appendix. Figure 17 indicates that the overall accuracy of the model trained by Faster R-CNN is higher than that of model trained by SSD. Figure 18 indicates that model trained by Faster R-CNN outperforms SSD in each fruit category.

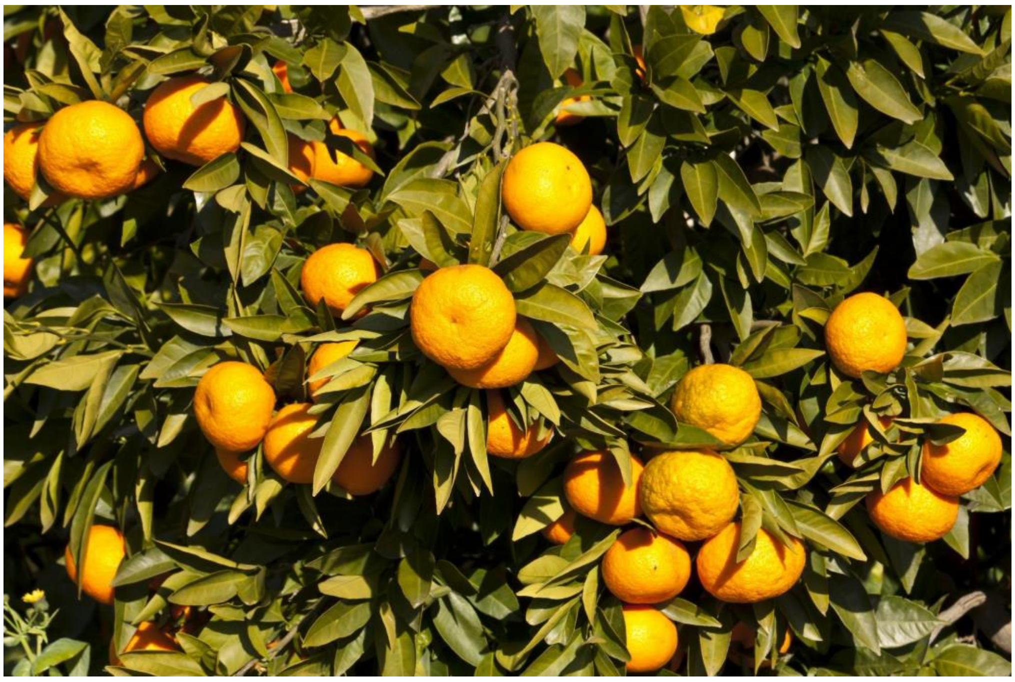


Figure 16 Example of test image

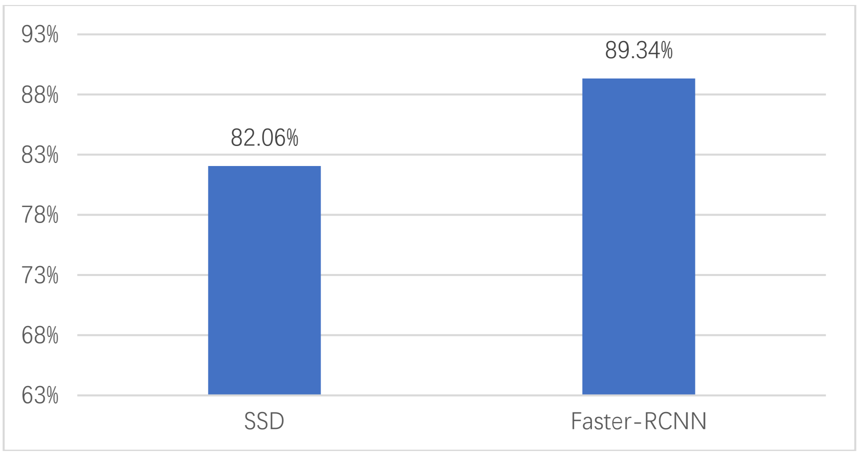


Figure 17 Overall model accuracy of SSD and Faster R-CNN

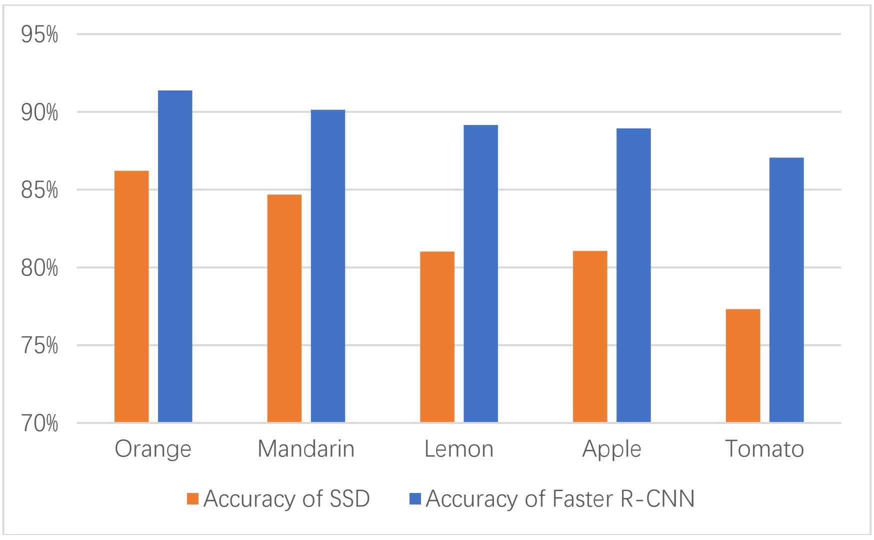


Figure 18 Accuracy of different fruits

In order to compare the result of these two models, two test images are used to illustrate model performance. The model trained by Faster R-CNN is better for small object detections compared with SSD, which could be seen in Figure 19 that the upper right tomato is not recognized by SSD. Faster R-CNN is also good at recognize objects obscured by other objects (i.e. leaves, branches, fruits), which is shown in Figure 20 that the apples obscured by leaves are recognized by Faster R-CNN rather than SSD. Since Faster R-CNN model first generates region proposals by selective search, then use CNN-based network to detect the object of each proposal, this two-shot approach makes it capable of detecting objects which is not easy to find from one single shot detecting, like SSD, the most representative one-stage object detection approach.

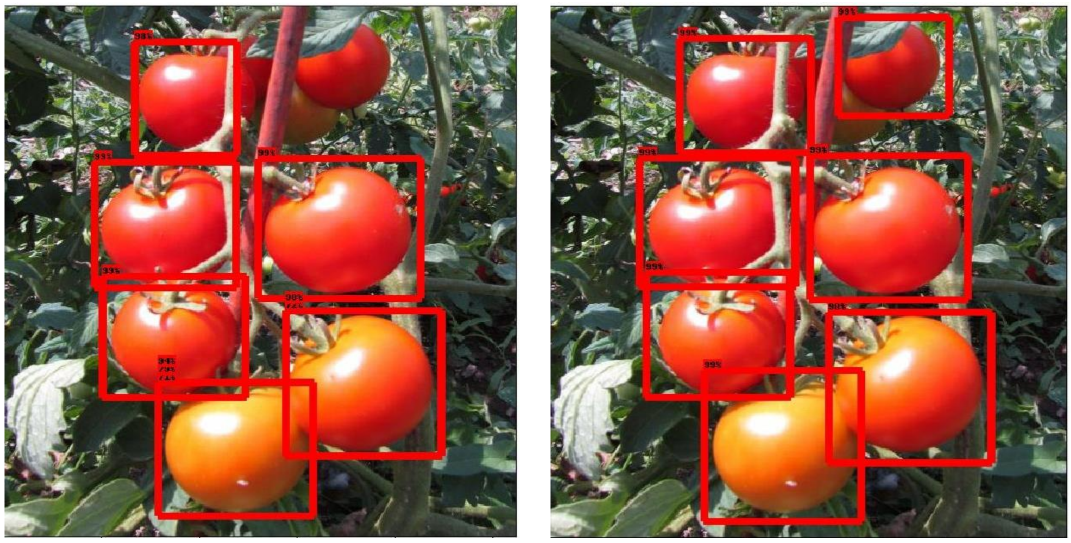

(a) Fruit detection by SSD (b) Fruit detection by Faster R-CNN

Figure 19 An example of fruit detection

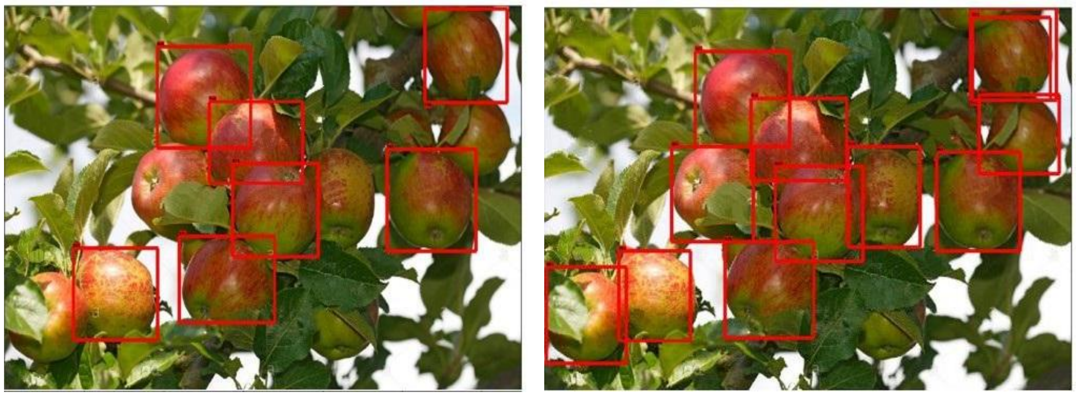

(a) Fruit detection by SSD (b) Fruit detection by Faster R-CNN

Figure 20 Another example of fruit detection

3.1.2. Speed Analysis

Figure 21 shows the average speed of the trained model processing an image in seconds. Obviously Faster R-CNN is slower than SSD, but the speed is still acceptable. Due to the difference of algorithms, since SSD only takes one shot of the whole image, there is no region proposal generation part in SSD, which can save time. However, the accuracy of one-stage model is often lower than two-stage model, which is also proved in our project. In object detection, to ensure the quality of model, speed is not the highest priority compared with accuracy generally.

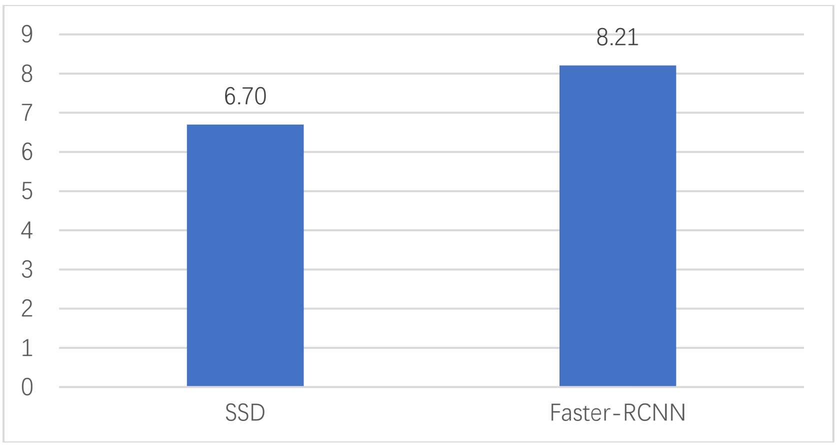


Figure 21 Second per image of SSD and Faster R-CNN

3.1.3. Loss Analysis

During training, TensorBoard could provide a real-time visualization of loss function to help monitor the performance of model. Loss function measures the performance of a classification model whose output is a probability value between 0 and 1, and gives a summary of how the training is progressing [9]. Loss increases as the predicted probability diverges from the actual label. Therefore, a perfect model should have a loss of 0.

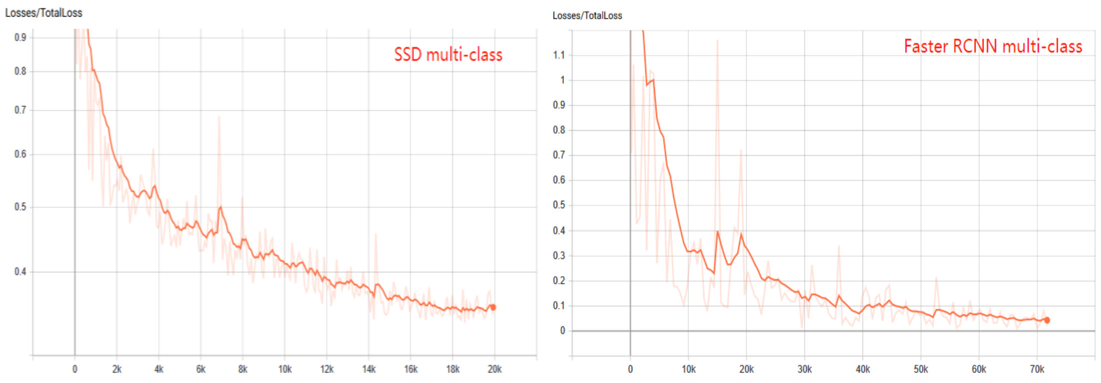


Figure 22 Loss function comparison

The loss of trained SSD and Faster R-CNN models both are decreasing slowly as the training progresses, and converge to a value close to 0, as shown in Figure 22, which indicates the models trained are successful and able to detect and classify objects. Obviously, the loss of Faster RCNN is less than that of SSD, which also proves that the accuracy of faster RCNN model is better than SSD from the mathematical perspective.

3.1.4. Overall Performance

Combining the analysis of accuracy, speed and loss of the trained SSD model and Faster R-CNN model, the accuracy of Faster R-CNN is significantly higher than SSD in each fruit categories. Although the processing speed of Faster R-CNN is slightly slower than SSD, the accuracy is the most important and primary evaluation metrics in this project. Therefore, the overall performance of Faster R-CNN is better than SSD, and the model trained by Faster R-CNN is used in final application.

3.2. Common Detection Issues

3.2.1. Influence of Unbalanced Dataset

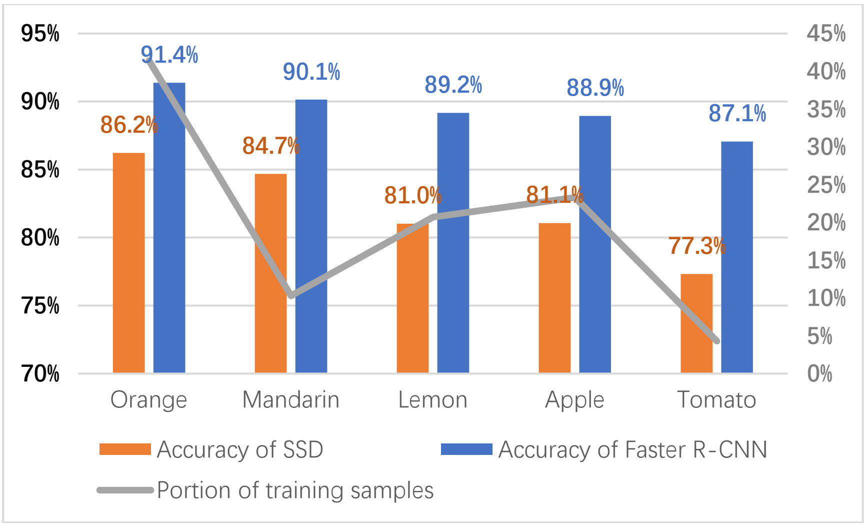


Figure 23 Relationship between accuracy and data volume

From the previous analysis, the accuracy of Faster R-CNN is higher than SSD in each fruit categories. However, as shown is Figure 23, it is obvious that orange has the highest recognition accuracy both in SSD and Faster R-CNN with the largest portion of training samples, while tomato has the lowest recognition accuracy both in SSD and Faster R-CNN with the smallest portion of training samples. It's notable that mandarin has the second highest accuracy but with small training data volume. After analysing the training dataset and test set, although the size of mandarin training set is small, most of the training images contain many mandarins, resulting in more labels than other categories with small portion of training samples. Besides, the colour difference between mandarin and leaf makes it easier to detect mandarins. Therefore, except the error in training dataset, the relationship between accuracy and training data volume follows the rule of accuracy decreases along with the decrease of the portion of training samples generally.

This phenomenon indicates a popular discussed issue that unbalance dataset may reduce the recognition accuracy of category with less data volume [13]. To avoid this issue, the training set should be well balanced with sufficient training data [10].

3.2.2. Underestimation

a) Input with Low Quality Images

With limited image resolution or terrible natural illumination conditions (i.e. overexposure, overshadowing), the features of fruits would be ambiguous, and the edges of fruits would be hard to identify. Under this circumstance, model would have unsatisfied performance. To ensure model accuracy, both training images and testing images should have high resolution and suitable natural illumination.

b) Fruit Cluster

Fruit clusters commonly appear in real-word orchard, which is also inevitable for every fruit counting system to deal with. In our project, the trained model could almost find all fruits which are clear and not heavily obscured by other objects, as shown in Figure 24, but still unable to detect the heavily obscured fruits in fruit clusters [11]. One reason is that when doing labelling part, the heavily overlapped fruits are not taken into account as we consider they are unclear and useless for training, therefore there is no heavily overlapped fruits in training samples. One possible solution is to use simple heuristics based on shape and size [12].

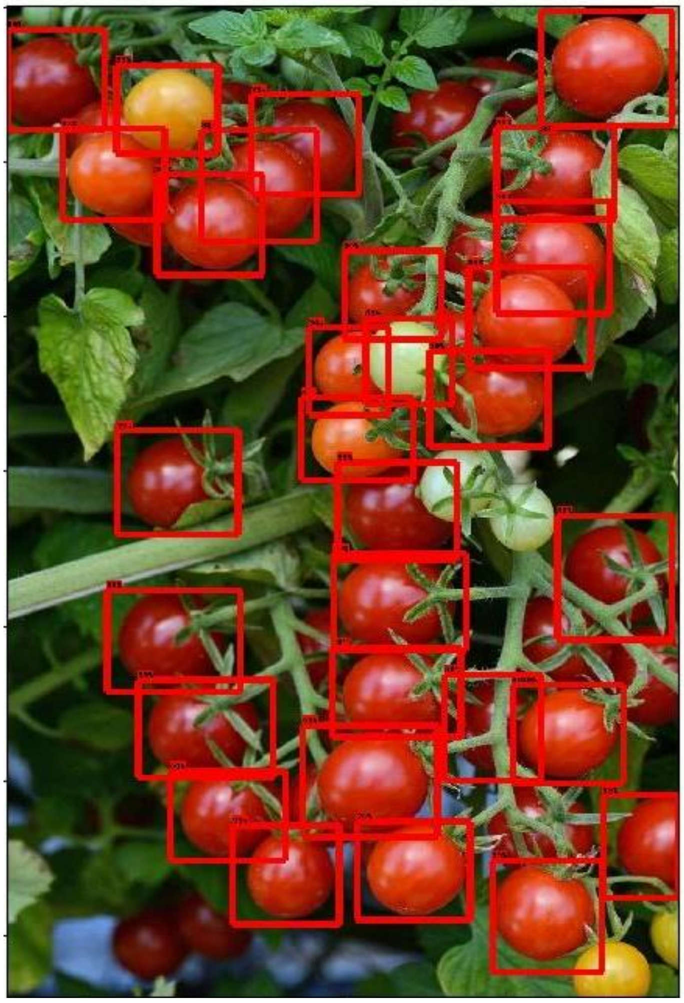


Figure 24 Counting result of fruit cluster

3.2.3. Overestimation

a) Identifying Leaf as Fruit

Some round shaped or oval shaped leaves may be identified as green oranges or limes with the effect of natural illumination and image resolution, as shown in Figure 25. One way to avoid this problem is the augmentation of training sample with high quality to help model to learn the features of fruits.

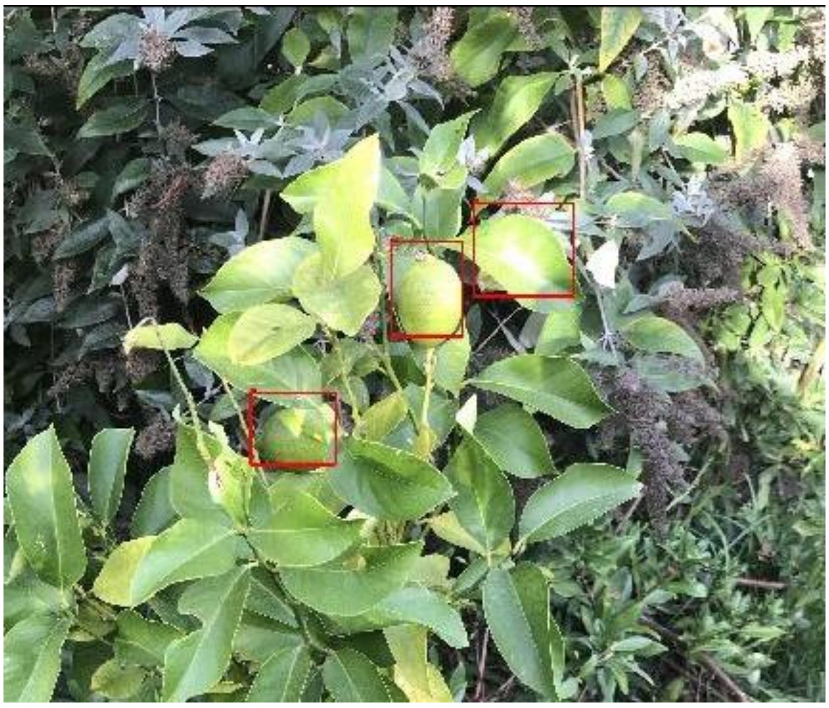


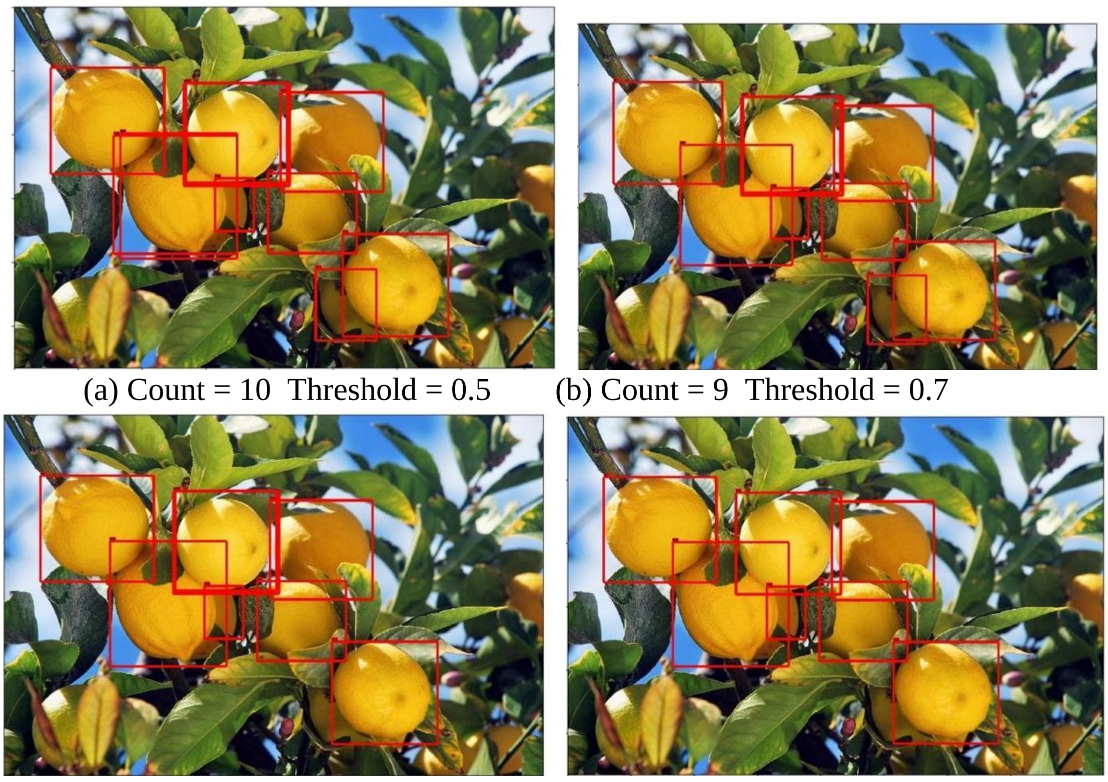
Figure 25 Example of identifying leaves as fruit

b) Underfitting

Since the objective of this project is not doing classification, the category of the model identifies doesn't matter, like identifying green round apple as green orange or lime is acceptable. However, misclassification sometimes would cause a problem in result in this project.

The training dataset we used consists of oranges, lemon, apple, mandarin and tomato, which have common features like round shape and similar colours. Most of the fruits in test images could be identified correctly with the confident score over 90%, as long as the fruits are clear. Due to the similarity of fruits, some fruits or part of the fruit may be identified as another kind of fruit simultaneously, so one fruit may be labelled with two boxes in the result. This problem could be fixed by adjust the confident score threshold, as shown in Figure 25. Increasing the threshold could filter the objects with lower confident score and improve the probability each object identified.
However, underfitting is not the problem we want to totally avoid. Underfitting could help model to identify the fruits which are obscured by leaves or fruits to a certain extent [14]. As shown in Figure 26 (b) and (c), when increasing confident score threshold from 0.7 to 0.8, the bottom right lemon is missed since it is obscured and dark, the confident score of this lemon is less than 0.8. After experiments whose results are shown in Figure 27, the confident score threshold of 0.7 has the highest accuracy and is used in final

application.


(a) Count = 10 Threshold = 0.5   (b) Count = 9 Threshold = 0.7

(c) Count = 8 Threshold = 0.8   (d) Count = 7 Threshold = 0.9

Figure 26 Example of Parameter adjustment

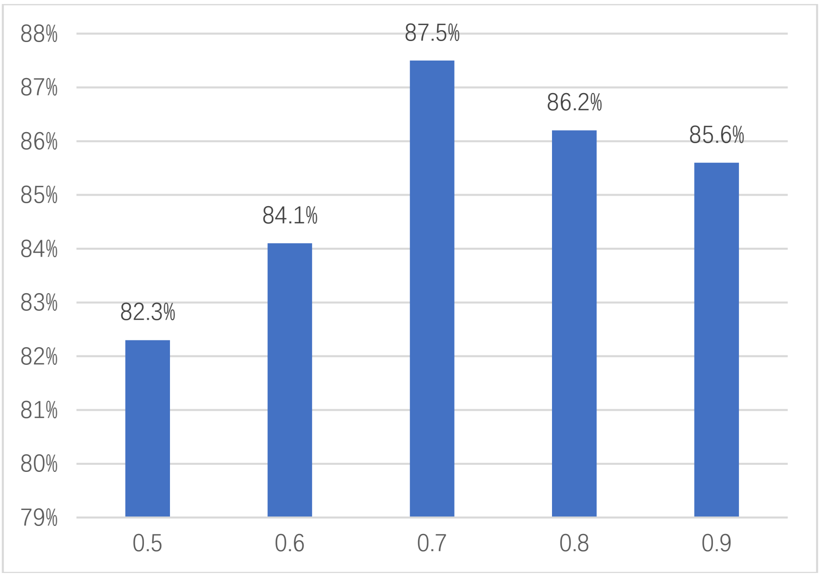


Figure 27 Accuracy with different thresholds

Except adjusting parameters, it could also be alleviated by improving the quality of training data. More training samples with high quality should be used to help machine understand and learn the features of each category. Since labelling training data is based on the human judgement, labellers should reach the consensus of the definitions of fruits and maintain the consistency during labelling [15].

3.3. Improvements

3.3.1. Small Object Detection

Small object detection still remains challenging in academic fields and is also a research hotspot. The state-of-the-art object detection models like Faster R-CNN has unsatisfactory performance as applied to detect small objects in images [16]. There are much fewer pixels available for small objects, which means weaker signal for the detector to utilize. The common way is to increase the feature resolution of small objects by magnifying the input images, which often results in heavy time consumption for training and testing.

Due to this, our model also has difficulty in counting fruits with the input image of the whole tree. Figure 28 shows the system results with the input of the whole tree. Since fruits are small, in (a) there are 13 fruits counted by model, and in (b) there are only 2 fruits counted by model.

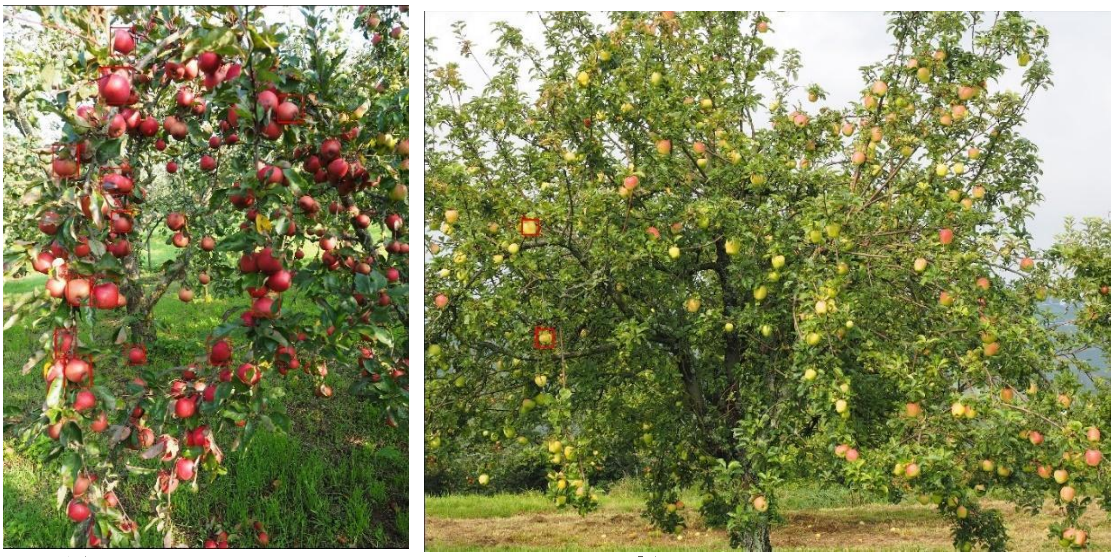

(a) System count is 13 (b) System count is 2

Figure 28 Results of small object detection

One solution is to augment the training dataset with small fruits. However, it would make model hard to identify the difference of different kinds of fruits, or even maybe increase the noise in training data and have side effects. Another feasible solution is to cut the image into several pieces before running in model [17]. The input image must have high resolution to ensure the quality of the sub-images after cutting. The total number of fruits is the sum of each counting result.

To improve the performance, we cut the image into 4 pieces and 9 pieces, and ran these sub-images in the model to see the performance. As shown in Figure 29 that there are 71 fruits are recognized after cutting in 4 pieces, and 92 fruits are recognized after cutting in 9 pieces. Another example of cutting is shown in Figure 30 that there are 43 fruits are recognized after cutting in 4 pieces, and 90 fruits are recognized after cutting in 9 pieces. The experiment result proves that cutting images before running is effective for small object detection, and cutting image in 9 pieces performs better than cutting in 4 pieces, since small object becomes larger in sub-images which makes model easier to recognize.

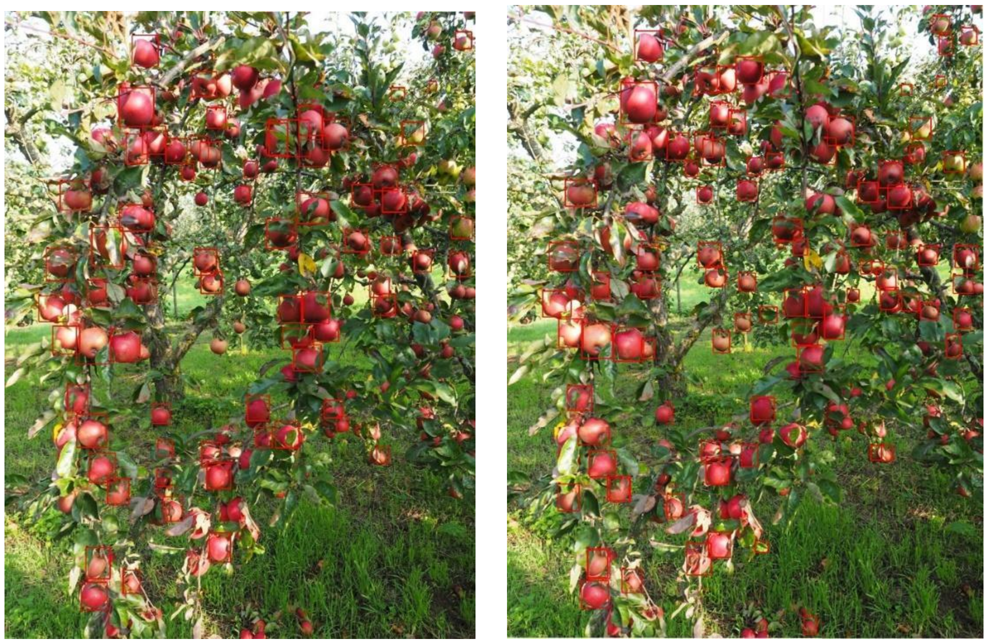

(a) Cutting into 4 pieces, count = 71 (b) Cutting into 9 pieces, count = 92

Figure 29 An example of improving small object detection by cutting

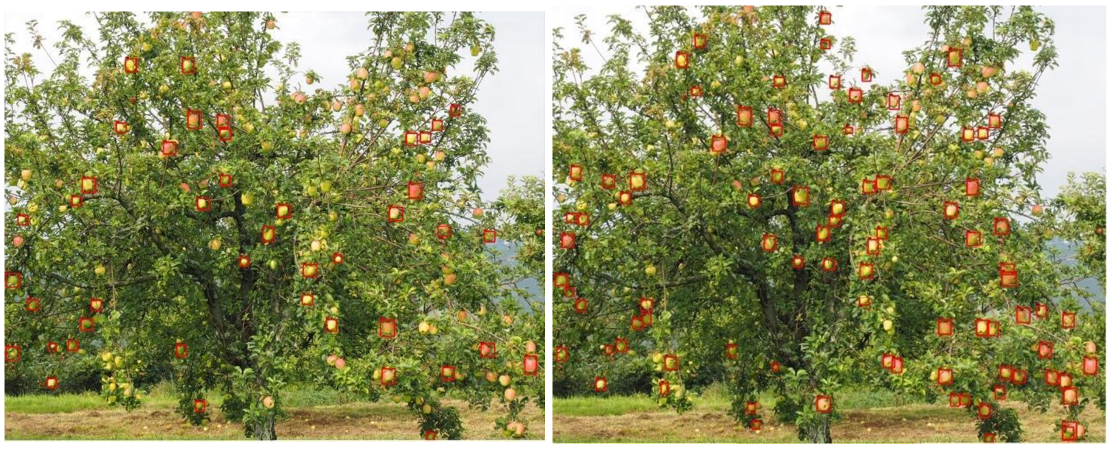

(a) Cutting into 4 pieces, count = 43 (b) Cutting into 9 pieces, count = 90

Figure 30 Another example of improving small object detection by cutting

3.3.2. Fruit Classification

The objective of this project is counting the number of fruits, and it is not necessary to identify the type of fruits. However, this functionality could be added into the system, as the classification of fruits will be indicated by the label and the colour of the box, as shown in Figure 31. Due to the similarity of fruits in training examples, some fruits may be misclassified, like lime may be classified as green orange, and red apple may be classified as tomato. Since the misclassification won't influence the counting result, the classification result could be used as an additional option.

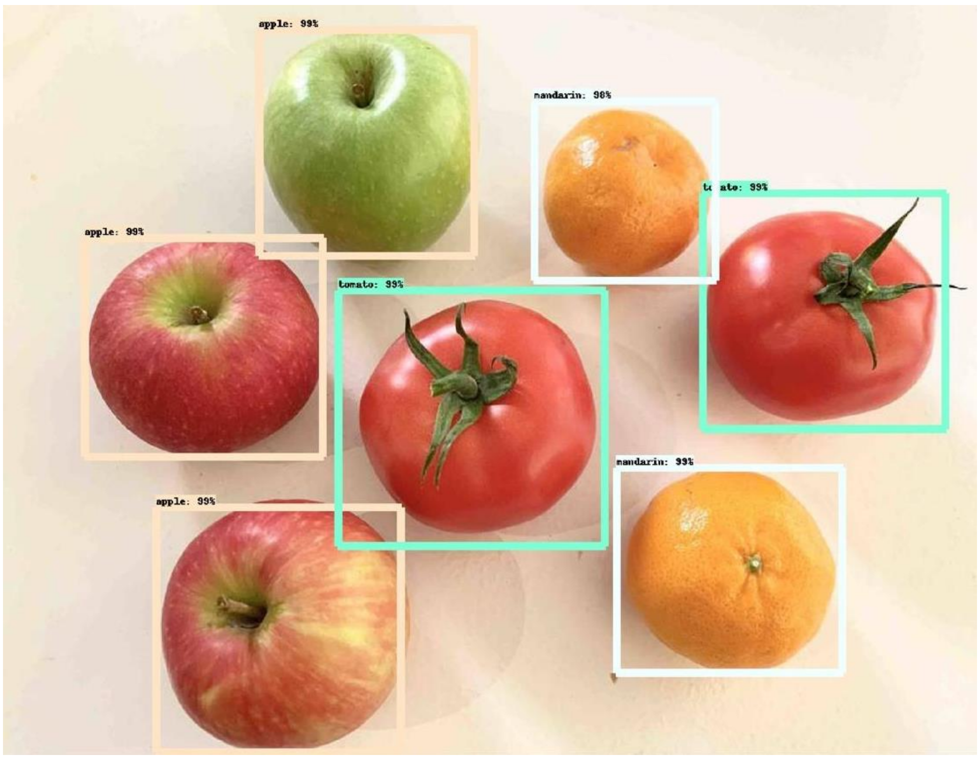


Figure 31 Fruit classification result

4 Application

In this project, we aim to develop an application for growers to estimate the fruit, i.e. orange, production of their trees. In the previous sections, we have discussed multiple models and methodologies to increase the accuracy on fruit detection. This section will be discussing how to make them work in practice.

4.1. Design Choice

Our training and evaluating processes are mainly finished on a desktop computer with a discrete graphics card which has a high-performance GPU to process the images. The actual performance, i.e. detecting speed, is quite well. However, we need to design a mobile application for our client use on field where we can capture the fruit image and get the result seamlessly. On a mobile platform, such a task may require more computing power to perform. At the beginning stage, we propose two solutions that can meet the requirement. The first one is that we can build a native mobile application on Android or iOS platform. The second one is to develop a web application that can serve on both mobile platforms and even on a PC.

4.1.1. Native Mobile App

Mobile apps provide smooth experience for work, entertainment, and communication on smartphones. To meet this requirement, our model has to be loaded into smartphones where the mobile app can run it locally. Current practices based on our research show that a real-time video processing functionality is preferable to display the labelled results as the user moves the camera to capture different objects [5,7,18]. TensorFlow also provides a simplified version, TensorFlow Lite, for mobile and IoT deployment. Our pre-trained model can be converted into mobile-friendly version that fits the performance of mobile devices.

4.1.2. Web App

Web app is a kind of applications that runs on client-server model where user can access it by visiting the site via a web browser. For instance, Google is a web app that almost everyone uses every day. It requires less resources on the client side and provides very similar experience to the users no matter what devices they use. Most web apps are accessible using web browsers. However, apart from the convenience on the client side, it requires more resources on the server side since most computational tasks are completed on the server. As the nature of web app indicated, it also requires network connection which assures communication between client and server. The complete TensorFlow framework can be deployed on any machine, i.e. Windows, macOS, or Linux, which works as a server. The detecting speed will be highly dependent on the hardware performance of the server.

We finally decide to develop a web application based on the project requirement and our previous web development experience, as shown in Figure 32. As mentioned above, our target users are fruit growers who need to estimate the fruit production by taking pictures of their trees. A native mobile app allows users to capture real-time footage which would be more intuitive. However, after we have finished data collecting and model training, we discover that the small objects could not be easily detected by our model. We assume that the actual performance of real-time video processing would be relatively weaker than the static image processing. In the actual use case, users may be standing at varying distances and capturing different parts of the tree. Therefore, the real-time video quality as well as the count would not be guaranteed. A static image processing solution should be more suitable to meet our client needs.

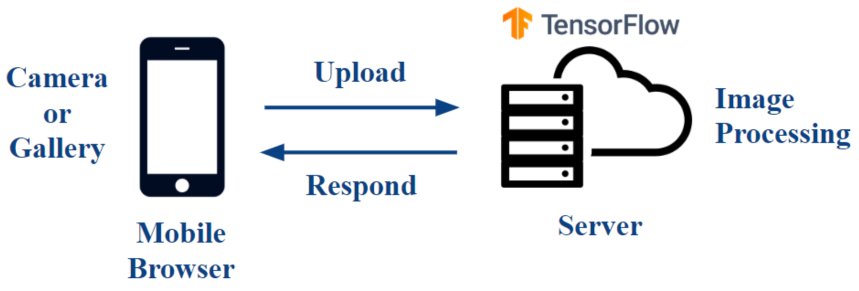


Figure 32 Client-server architecture

4.2. Implementation Details

4.2.1. Server

Our dataset pre-processing and model training are mainly finished in Python environment. To facilitate the process, the server backend is also implemented in Python. Flask framework is employed to set up a web server to route the connections and serve web pages as well as the static files. Flask is a Python framework that depends on two major components, Werkzeug WSGI toolkit and Jinja template engine. WSGI is a specification for Python web interface between servers and web apps proposed in 2003. [19] The object detection library from TensorFlow is also embedded in our server code. Our final model is trained based on faster_rcnn_inception_v2_coco. [20] When the server gets a POST request, it will first fetch and save the blob sent by Ajax function from the client. The second task, image processing, is performed once the image has been saved in local storage. We mainly use the code snippets provided in TensorFlow object detection tutorial Jupyter notebook to process the image using our final model. The labelled image is also saved in local storage, converted to a Base64 binary data string, and sent back to client as a part of JSON.
The Flask framework is not designed to run as a production server. For further use, a standalone WSGI web server like Gunicorn and a reverse proxy like Nginx should be deployed with Flask apps.

4.2.2. Client

The application on the client side is mainly developed in HTML and JavaScript which can be rendered in mobile browsers. The basic structure of the web app is included in a HTML file. The main functionalities employed are the input tag and canvas tag introduced by HTML5 standard. The input tag with an image file type allows the web app to call the OS specified functions, e.g. open camera or photo gallery. After importing the image, it will be processed by canvas API to fit the screen display and meet the TensorFlow requirement on the server side, i.e. pixel-wise image compression. The above step is manipulated by JavaScript code and handled by JavaScript engine of the browser. The captured or imported image can then be uploaded to the server by invoking an asynchronous file upload request, i.e. Ajax. The jQuery library is included to facilitate this process. As described above, the response will be parsed once it arrives at the client side. The labelled image is rendered on screen by creating a new canvas based on the responded Base64 binary data string.

4.3. User Interface

Since our target users are fruit growers, they may not have much experience on using complex mobile applications. To facilitate the use, we designed an intuitive user interface of our app on client side. As shown in Figure 33, the main page of the application consists of three components, i.e. header, canvas, and button. The header is the name of our app, Fruit Counter. The canvas section is to display the captured image or picture selected from phone gallery. The count and labelled result will be also displayed in this section. There are only two buttons available in the app, camera button and upload button. As indicated by name, user can click on the camera button to select the image source, capturing via camera or importing from gallery. After doing so, the upload button will be active to let user send the picture to the server for further processing. During the processing, a loading initiator will be displayed on screen. The result then will be displayed on the position where the previous image was shown. The user can easily check the labelled image by two-finger zooming to see the confidence score of each labelled fruit.

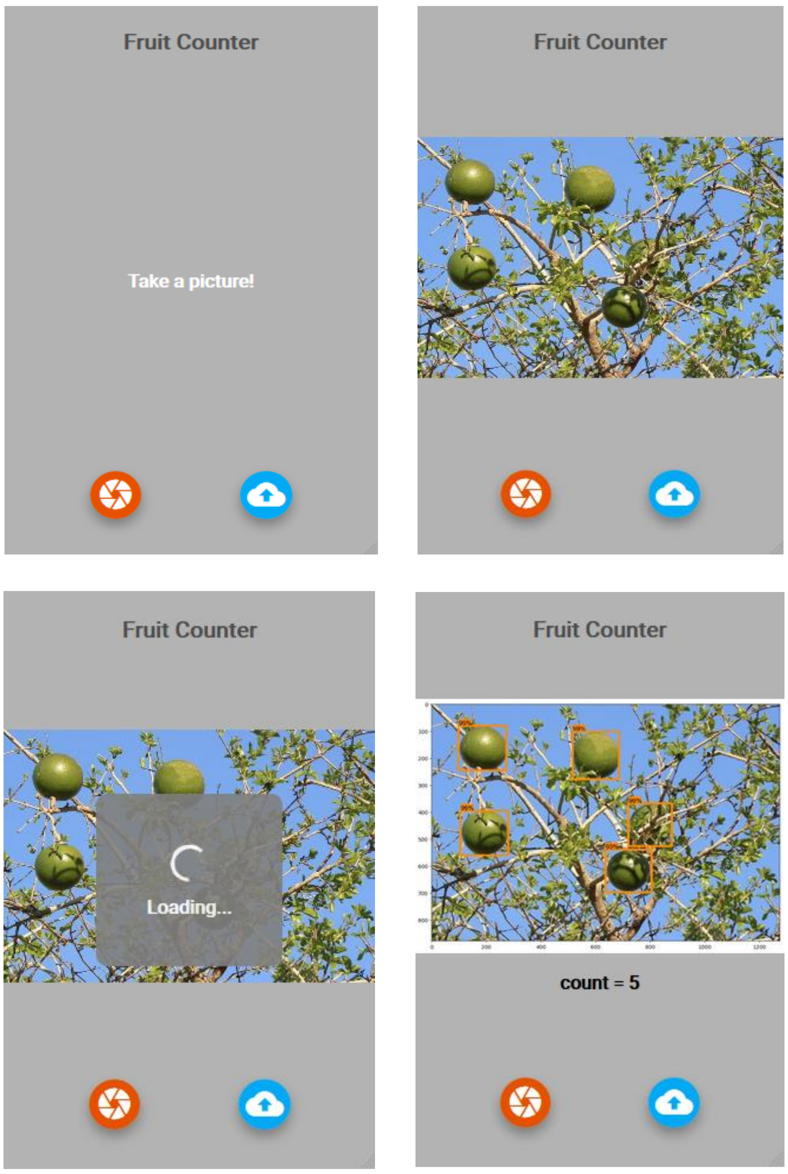


Figure 33 Screenshots of the web application

5 Future Work

5.1. Dataset Construction

The training data we collected is sufficient to make the model learn round-shaped fruits as required. However, as discussed in section 3.2, our model still does not perform to expectations in some cases. For instance, a rounded green leaf may be identified as a green orange or a lime. Zhu, Vondrick, et al. discovered that additional training data may not help the model recognize the features of an object. [21] But "clean" dataset may reduce the number of images we need to train. Besides the quantity, image quality and effectiveness are the critical part to consider. Special features of the fruit should be identified and trained with. Using more classes to label the same fruit under different conditions may also facilitate learning process as we manually label them according to specific features. Negative samples can also be included in the training data, e.g. the leaves in the background, which can be omitted later in actual use.

5.2. Small Object Detection

We have found that our model does not perform well when the fruits are too small in images. Image cropping has been discussed in section 3.3.1. To further improve the performance, Li, Jianan, et al. proposed a Perceptual Generative Adversarial network (GAN) to generate super-resolved representation for the small objects. [22] This method largely enhances the low-level features which effectively increase the detection accuracy. Some other approaches [16, 23] show proposal patches and context information are important in such training. These approaches mainly focus on how to resolve the issue due to the insufficiency of the low-level features of small objects in low resolution images.

5.3. Mobile App

As mentioned in section 4.1.1, a native mobile app built for specific platform is more preferable. The pre-trained model can be embedded on the client side, which keeps the image processing pipeline completely on the phone without network connection. The latency due to network transmission can be minimized by employing TensorFlow Lite. On the other hand, web apps will be developed rapidly in 5G era. TensorFlow also showed its support to web development, i.e. TensorFlow.js library. High resolution and real-time image processing will require less resources and largely facilitate the detecting process on the mobile side in the near future.

6 Conclusion

We applied transfer learning on pre-trained model to develop an object detection application for fruit growers to estimate the production of fruit trees. As discussed at the beginning, we first conducted research on current object detection models and algorithms. According to the models we had selected, i.e. SSD and Faster R-CNN, the training and testing pipeline was developed to perform transfer learning with pre-trained models provided by TensorFlow based on COCO dataset. We collected the fruit images from the internet for training purposes. After the first evaluation, we discovered the quantity of our training images was inadequate. Multiple classes of fruits, e.g. lime, tomato, apple, were included in the training set. During the analysis, we found out that our trained models underperform when the high-level features of detecting object and background are similar. Small object detection was also hard to perform due to the insufficient low-level features retained on the compressed images. To remedy this situation, we cropped the original one into four and processed them independently to make the small objects larger. However, the objects lain on the edges of the cropped images may be omitted. Therefore, improved image processing algorithms are needed. Finally, a web app was developed with the final model based on Faster R-CNN which achieved 89.34% average accuracy.

Appendix A: Links

Source code:
https://github.com/Kartoro/COMP90055-FruitDetection
Demo video link:
https://www.youtube.com/watch?v=JXQLtP5A_uI

Appendix B: Test Records

Tables of test records:

Image	Manual Count	System Count	Differenc e	Accuracy
Orange1	18	16	2	88.9%
Orange2	12	10	2	83.3%
Orange3	11	9	2	81.8%
Orange4	17	15	2	88.2%
Orange5	9	8	1	88.9%
Apple1	12	10	2	83.3%
Apple2	22	18	4	81.8%
Apple3	16	13	3	87.5%
Apple4	20	16	4	80%
Apple5	11	8	3	72.7%
Tomato1	9	8	1	88.9%
Tomato2	33	25	8	75.7%
Tomato3	13	10	3	76.9%
Tomato4	29	24	5	82.8%
Tomato5	13	9	4	62.3%
Mandarin1	15	12	3	80%
Mandarin2	26	23	3	88.5%
Mandarin3	15	13	2	87%
Mandarin4	26	22	4	84.6%
Mandarin5	12	10	2	83.3%
Lemon1	13	9	4	62.3%
Lemon2	24	20	4	83.3%
Lemon3	15	13	2	87%
Lemon4	20	17	3	85%
Lemon5	16	14	2	87.5%
				Average accuracy = 82.06%

Table 1. Test records of trained SSD model

Image	Manual Count	System Count	Differenc e	Accuracy
Orange1	18	16	2	88.9%
Orange2	12	12	0	100%
Orange3	11	10	1	90.9%
Orange4	17	15	2	88.2%
Orange5	9	10	1	88.9%
Apple1	12	11	1	91.7%
Apple2	22	20	2	90.9%
Apple3	16	13	3	81.2%
Apple4	20	18	2	90%
Apple5	11	10	1	90.9%
Tomato1	9	11	2	77.8%
Tomato2	33	30	3	90.9%
Tomato3	13	15	2	84.6%
Tomato4	29	26	3	89.7%
Tomato5	13	12	1	92.3%
Mandarin1	15	14	1	93.3%
Mandarin2	26	23	3	88.5%
Mandarin3	15	14	1	93.3%
Mandarin4	26	24	2	92.3%
Mandarin5	12	10	2	83.3%
Lemon1	13	11	2	84.6%
Lemon2	24	22	2	91.7%
Lemon3	15	13	2	87%
Lemon4	20	19	1	95%
Lemon5	16	14	2	87.5%
				Average accuracy = 89.34%

Table 2. Test records of trained Faster R-CNN model

Appendix C: Use Case

A simple use case can be described as following:
The user

1. Stands about two meters away from an orange tree;
2. Takes out the smartphone and opens a browser, e.g. Chrome, Safari, Firefox;
3. Types in the web address and visits the site;
4. Waits until the browser finished rendering the web pages;
5. Clicks on the orange ‘camera’ button on the page;
6. Selects ‘camera’;
7. Holds the phone steadily and makes the camera focus on the target tree;
8. Clicks on the capture button and finishes capturing;
9. Reviews the picture shown on the page;
10. Clicks on the blue ‘upload’ button;
11. Waits until the result sent back by server (loading indicator);
12. Reviews the count and labelled image displayed on the screen.

References

[1] Liang, Hong & Sun, Xiao & Yunlei, Sun & Gao, Yuan. (2017). Text feature extraction based on deep learning: a review. EURASIP Journal on Wireless Communications and Networking. 2017.

[2] Palmer, R., Borck, M., West, G., & Tan, T. (2012). INTENSITY AND RANGE IMAGE BASED FEATURES FOR OBJECT DETECTION IN MOBILE MAPPING DATA.

[3] Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1717-1724).

[4] Oliva, A., & Torralba, A. (2007). The role of context in object recognition. Trends in cognitive sciences, 11(12), 520-527.

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

[6] Jiang, H., & Learned-Miller, E. (2017, May). Face detection with the faster R-CNN. In 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017) (pp. 650-657). IEEE.

[7] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[8] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

[9] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186).

[10] Chen, H., Xiong, F., Wu, D., Zheng, L., Peng, A. (2017). Assessing impacts of data volume and data set balance in using deep learning approach to human activity recognition. IEEE International Conference on Bioinformatics & Biomedicine. IEEE Computer Society.

[11] Mohammed, F.G..; Amer, W.A.. Color-Based for Tree Yield Fruits Image Counting. Preprints 2017.

[12] Chen, S. W., Shivakumar, S. S., Dcunha, S., Das, J., Okon, E., Qu, C., ... & Kumar, V. (2017). Counting apples and oranges with deep learning: A data-driven approach. IEEE Robotics and Automation Letters, 2(2), 781-788.

[13] Chen, H., Xiong, F., Wu, D., Zheng, L., Peng, A., Hong, X., ... & Zheng, H. (2017, November). Assessing impacts of data volume and data set balance in using deep learning approach to human activity recognition. In 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 1160-1165).

[14] Bullinaria, J. A. (2015). Bias and Variance, Under-Fitting and Over-Fitting.

[15] Bargoti, S. (2017). Fruit Detection and Tree Segmentation for Yield Mapping in Orchards.

[16] Chen, C., Liu, M. Y., Tuzel, O., & Xiao, J. (2016, November). R-CNN for small object detection. In Asian conference on computer vision (pp. 214-230). Springer, Cham.

[17] Stein, M., Bargoti, S., & Underwood, J. (2016). Image based mango fruit detection, localisation and yield estimation using multiple view geometry. Sensors, 16(11), 1915.

[18] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).

[19] Eby, Phillip J. (2003). PEP 333 -- Python Web Server Gateway Interface v1.0. Python Software Foundation. Retrieved from https://www.python.org/dev/peps/pep-0333/

[20] TensorFlow. (2018). TensorFlow Object Detection API. Retrieved from https://github.com/tensorflow/models/tree/master/research/object_detection

[21] Zhu, X., Vondrick, C., Ramanan, D., & Fowlkes, C. C. (2012, September). Do We Need More Training Data or Better Models for Object Detection? In BMVC (Vol. 3, p. 5).

[22] Li, J., Liang, X., Wei, Y., Xu, T., Feng, J., & Yan, S. (2017). Perceptual generative adversarial networks for small object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1222-1230).

[23] Bargoti, S., & Underwood, J. (2017, May). Deep fruit detection in orchards. In 2017 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3626-3633). IEEE.