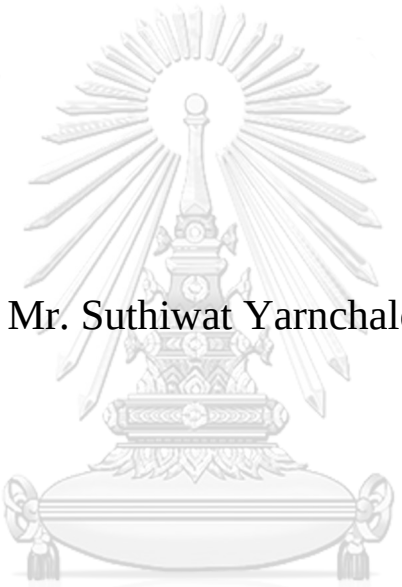
REAL-TIME INSTANCE SEGMENTATION AND POINT CLOUD EXTRACTION FOR JAPANESE FOOD USING RGB-D CAMERA

Mr. Suthiwat Yarnchalothorn



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Cyber-Physical System
Department of Mechanical Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2020

การตรวจจับวัตถุในระดับพิกเซลแบบทันทีและการสกัดพิกัดสามมิติสำหรับอาหารญี่ปุ่นโดยใช้กล้อง RGB-D

นายสุทิวัส ญาณชโลทร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาระบบกายภาพที่เชื่อมประสานด้วยเครือข่ายไซเบอร์ ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2563

Thesis Title	REAL-TIME INSTANCE SEGMENTATION AND POINT CLOUD EXTRACTION FOR JAPANESE FOOD USING RGB-D CAMERA
By	Mr. Suthiwat Yarnchalothorn
Field of Study	Cyber-Physical System
Thesis Advisor	NATTAPOL DAMRONGPLASIT, Ph.D.
Thesis Co Advisor	Professor Hayashi Eiji, Ph.D.

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

.....
Dean of the FACULTY OF ENGINEERING
(Professor SUPOT TEACHAVORASINSKUN)

THESIS COMMITTEE

.....
Chairman
(Professor PAIROD SINGHATANADGID, Ph.D.)

.....
Thesis Advisor
(NATTAPOL DAMRONGPLASIT, Ph.D.)

.....
Thesis Co-Advisor
(Professor Hayashi Eiji, Ph.D.)

.....
Examiner
(Associate Professor ALONGKORN PIMPIN, Ph.D.)

.....
External Examiner
(Assistant Professor Kakanand Srungboonmee, Ph.D.)

สุทิวัส ญาณชโลทร : การตรวจจับวัตถุในระดับพิกเซลแบบทันทีและการสกัดพิกัดสามมิติสำหรับอาหารญี่ปุ่นโดยใช้กล้อง RGB-D. (REAL-TIME INSTANCE SEGMENTATION AND POINT CLOUD EXTRACTION FOR JAPANESE FOOD USING RGB-D CAMERA) อ.ที่ปรึกษาหลัก : ดร.ณัฐพล ดำรงค์พลาสิทธิ์, อ.ที่ปรึกษาร่วม : ศ. ดร.ฮายาชิ อิจิ

ในปัจจุบันนวัตกรรมส่งผลให้เกิดการพัฒนาอุตสาหกรรมอาหาร สังเกตได้จากความนิยมที่เพิ่มขึ้นของการวิจารณ์อาหารบนอินเตอร์เน็ตและธุรกิจการจัดส่งอาหารแบบรวดเร็ว ในทำนองเดียวกันกระบวนการผลิตและกระบวนการบรรจุอาหารใส่บรรจุภัณฑ์จะเปลี่ยนจากใช้แรงงานคนเป็นอัตโนมัติโดยใช้หุ่นยนต์เข้ามาแทนที่อย่างแพร่หลาย การเปลี่ยนเปลงนี้จะทำให้ผู้ผลิตสามารถควบคุมคุณภาพอาหารและเพิ่มประสิทธิภาพในกระบวนการผลิตได้ อย่างไรก็ตามปัจจัยที่สำคัญอย่างหนึ่งที่จะทำให้สิ่งนี้เป็นไปได้คือความสามารถในการตรวจจับและแยกประเภทของอาหารจากภาพถ่ายอย่างแม่นยำด้วยความเร็วสูง

ในงานวิจัยนี้เราจะศึกษาการตรวจจับวัตถุอาหารแบบทันทีโดยใช้ภาพจากกล้องวัดความลึก วิธีที่เลือกใช้คือการตรวจจับวัตุในระดับพิกเซลโดยใช้การเรียนรู้แบบอัตโนมัติที่มีโครงข่ายประสาทหลายชั้นเพื่อตรวจจับชิ้นอาหารญี่ปุ่นในระดับพิกเซล ในที่นี้จะใช้แบบจำลอง 2 แบบ คือ Cascade Mask R-CNN และ Hybrid Task Cascade โดยแบบจำลองทั้งหมดจะเรียนรู้ด้วยตัวมันเองบนทั้งหมดสองแพลตฟอร์ม คือ บนเครื่องคอมพิวเตอร์ และบนบริการคลาวด์ จากนั้นได้ทำการศึกษาแบบจำลองที่สร้างขึ้นในสภาวะต่าง ๆ นอกจากนี้จะนำข้อมูลความลึกที่ได้จากกล้องมาประสานกับข้อมูลการตรวจจับวัตถุที่ได้จากขั้นตอนแรกเพื่อสกัดข้อมูลพิกัดสามมิติของวัตถุอาหารซึ่งจะสามารถนำมาใช้ประโยชน์ในกระบวนการผลิตอาหารแบบอัตโนมัติ เช่น การหยิบและวางชิ้นอาหารซึ่งมีรูปร่างและขนาดที่หลากหลายได้อย่างแม่นยำ

จากผลการทดลองพบว่าแบบจำลอง HTC มีความแม่นยำสูงกว่าแบบจำลอง Cascade Mask R-CNN บนทั้งสองแพลตฟอร์มที่ใช้ในการเรียนรู้อัตโนมัติ แต่ในทางกลับกันแบบจำลอง HTC จะมีความเร็วในการตรวจจับที่ช้ากว่า จากนั้นยังพบว่าความเร็วในการตรวจจับวัตถุของทั้งสองแบบจำลองมีแนวโน้มจะลดลงเมื่อจำนวนวัตถุในภาพเพิ่มขึ้นและเมื่อความละเอียดของภาพเพิ่มขึ้นยิ่งไปกว่านั้นผลการทดลองแสดงให้เห็นว่าการเปลี่ยนแปลงสภาพแวดล้อม ได้แก่ การเปลี่ยนสีพื้นหลัง การปรับลดความสว่าง การวางวัตถุอาหารซ้อนทับ และการใช้อาหารที่ไม่สมบูรณ์ ส่งผลให้ความแม่นยำของแบบจำลอง HTC ลดลง หลังจากนั้นได้ทำการสกัดพิกัดสามมิติของวัตถุอาหารออกมาโดยมีความเร็วเฉลี่ยอยู่ที่ 6.71 เฟรมต่อวินาที

สาขาวิชา	ระบบกายภาพที่เชื่อมประสานด้วยเครือข่ายไซเบอร์	ลายมือชื่อนิสิต .....
ปีการศึกษา	2563	ลายมือชื่อ อ.ที่ปรึกษาหลัก .....
		ลายมือชื่อ อ.ที่ปรึกษาร่วม .....

6270375221 : MAJOR CYBER-PHYSICAL SYSTEM
KEYWORD: Object detection, Instance segmentation, 3D point cloud, Food automation, Artificial Intelligence, Depth camera

Suthiwat Yarnchalothorn : REAL-TIME INSTANCE SEGMENTATION AND POINT CLOUD EXTRACTION FOR JAPANESE FOOD USING RGB-D CAMERA. Advisor: NATTAPOL DAMRONGPLASIT, Ph.D. Co-advisor: Prof. Hayashi Eiji, Ph.D.

Innovation in technology is playing an important role in the development of food industry, as is indicated by the growing number of food review and food delivery applications. Similarly, it is expected that the process of producing and packaging food itself will become increasingly automated using a robotic system. The shift towards food automation would help ensure quality control of food products and improve production line efficiency. One key enabler for such automated system is the ability to detect and classify food object with great accuracy and speed.

In this study, we explore real-time food object segmentation using RGB-D depth camera. Instance segmentation based on 2D RGB data is used to classify Japanese food objects at a pixel-level with Cascade Mask R-CNN and Hybrid Task Cascade deep learning models. The model is trained on both local GPU and cloud service. The precision and recall values for classifying food objects under different scenario conditions are investigated. Furthermore, we construct 3D point cloud of food objects using depth information from the camera, which will help facilitate food automation operation such as precision grasping of food object with numerous shapes and sizes.

The result shows that the trained HTC model has better precision than Cascade Mask R-CNN model, albeit at a lower detection speed. The inference speed of both models monotonically decreases as the number of food objects and image resolution of the processed image increase. In addition, it is found that that the accuracy of the HTC detection can be quite sensitive to environmental factors such as background colors, low brightness, and having an incomplete object. The 2D segmentation result is combined with 3D point cloud extraction to realize real-time 3D segmentation of Japanese food objects with an average framerate of 6.71 fps.

Field of Study:	Cyber-Physical System	Student's Signature .....
Academic Year:	2020	Advisor's Signature .....
		Co-advisor's Signature .....

ACKNOWLEDGEMENTS

This thesis has been completed with a lot of assistance from many people.

I would first like to acknowledge the Department of Mechanical Engineering, Chulalongkorn University for providing the financial support for my study and research.

I am grateful to Dr. Nattapol Damrongplasit, my advisor, who devoted his time to guide me, gave me intellectual support and encouragement, and provided English correction of my writings.

I would like to express my special thanks to Professor Hayashi Eiji who allowed me to be a member of Hayashi Laboratory and funded me during my research in Japan.

I would also like to thank other professors in Cyber-Physical System program for giving me advices for my research and future career.

Lastly, a thank you from the heart to all my friends in this program, members of Hayashi Lab, and all people who always supported me throughout my study.

Suthiwat Yarnchalothorn

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1 INTRODUCTION

1.1 Background

Food industry is an important part of the economy for many countries. It represents a large portion of the country's GDP, and it is made up of various business sectors that provide food supply to the population, including agriculture, manufacturing, food processing, and food services, just to name a few. By incorporating innovative technologies to these existing sectors, the overall efficiency can be improved, while the operating cost can be reduced. One such example is the use of automation in the food production process. Automating food production can help to improve consistency in the appearance and quality of the food being produced, as well as to minimize unnecessary waste by using minimum amount of ingredients. Although the benefit of automation in food production is clear, there has not been a wide-scale adoption of such technology in the food industry, as one might observe in other industries like car or industrial manufacturing. One reason has to do with the ability to manipulate the components of the object being produced. Unlike other industries where the components are usually uniform in size and weight, the ingredients that go into food production are often of diverse size, shape, weight, and texture, making it difficult to develop a line automation using traditional methods [1].

Computer vision may help solve these challenging problems, however. The technology mimics how human sees and perceives things. One important aspect of computer vision is image recognition, which involves different processes such as object detection, classification, and segmentation. A particular technique, called instance segmentation, has been used to detect distinct objects in an image in real-time and the classification is done at a pixel-level using deep learning model. Previous studies have shown this approach to be effective and reliable in detecting food objects [2-4].

In addition to the output RGB data from a camera sensor, depth camera technology can also provide 3D depth information which can be advantageous for analyzing 3D objects in a surrounding environment. By combining segmented RGB data and 3D depth information together, automatic detection and classification of food object can be done more accurately, allowing for better estimation of its shape, weight, size and, even calories.

In this study, we use the data from a single stereo depth camera mounted on a robot end-effector which is used for performing tasks involving food automation. The camera outputs are RGB color data and depth information. We perform instance segmentation on RGB data by using Cascade Mask R-CNN and HTC (without semantic segmentation) which are deep learning architectures based on mmdetection [5] benchmark results. The Cascade Mask R-CNN method is a combination of Cascade R-CNN [2] and Mask R-CNN [3], which allows classification of RGB data at a pixel level. Then, we combine the processed data with depth information to achieve 3D object segmentation. All of the output information from our proposed process is published on a robot system network that can later be used in path planning or grasping posture estimation. Moreover, the resulting output can be generalized to other systems as well, such as warehouse inventory tracking or human activity monitoring, not just limited to a robotic system.

1.2 Objectives and Scope

1.2.1 Objectives

1) Develop and apply instance segmentation model to detect Japanese food objects at a pixel-level.
2) Develop a point cloud extraction method of food objects in real-time using RGB-D camera

1.2.2 Scope

The experiment is performed using only one stereo depth camera mounted on a manipulator's end-effector or a stationary frame. The camera is connected to a PC equipped with a local GPU, or has access to Google Colab, to perform image processing tasks.

The Japanese food dataset will be curated and labeled by-hand in COCO-style. It will contain images taken from the web and those captured by the researcher.

Detection model for Japanese food will be developed to detect, classify, and segment food objects in real-time using RGB information. The refinement of the models will be accomplished through modifying and tuning of different

parameters such as datasets, training batch size, deep learning architectures, and activation functions. Different scenario condition that could impact the accuracy and robustness of the model will also be explored.

The RGB segmentation result will be fused with depth information from depth camera to extract point cloud of food objects in real-time. Inference speed will be evaluated for different extraction methods.

1.3 Timeline

Table 1: Thesis plan for 2020

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Task assigned												
Learn and Practice object detection												
Create datasets												
Train instance segmentation models												
Point cloud extraction												
Experiments												
Improve datasets												
Thesis paper												
Conference paper												
Proposal Exam												
SICE 2020												

1.4 Expected Outcome

- Gain a comprehensive understanding in regards to object detection and related methods in the field of computer vision.
- Establish an object detection framework that can detect Japanese food objects.
- Establish a module that can extract point cloud of food objects in real-time.
- Able to evaluate and compare accuracies between models, as well as to assess how different environmental conditions might impact those models.

CHAPTER 2 LITERATURE REVIEW

2.1 Object Detection and Instance Segmentation

Object Detection, Semantic Segmentation, and Instance Segmentation are some of the most popular and important fields in Computer Vision [6, 7]. Object Detection is a method of classifying and localizing all the objects in an image. It locates the exact positions of objects and labels them into classes. The position of an object is usually located by a bounding box, which is represented by a rectangular enclosing region. Semantic Segmentation is the process of identifying every single pixel in an image into a class, including objects and background such as sky and grass. Unlike object detection, this process only identifies pixels in an image. It does not consider a cluster of pixels as an object. Instance Segmentation, on the other hand, can be viewed as a combination of Object Detection and Semantic Segmentation. It assigns a class to each pixel and treats them as an object. Instance Segmentation also treats multiple objects of the same class as individual objects that have separate entities.

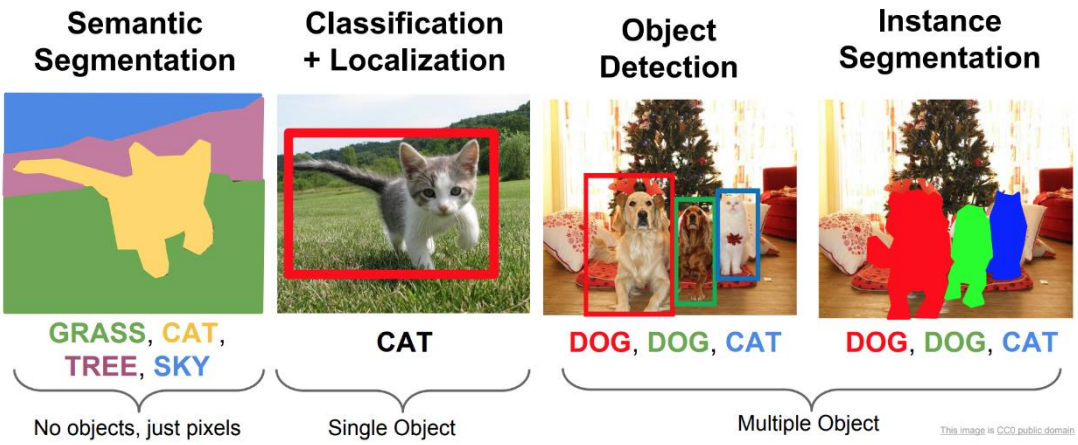


Fig. 1 Object Detection and its related methods

There are many essential concepts associated with Object Detection such as Bounding Box, Anchor Boxes, Intersection over Union (IoU). They will be explained in this section.

First, Bounding Box is a rectangular-shaped box that can be described by 4 parameters; bx, by, bw, and bh where (bx, by) is the center position of the box and bw and bh are the width and the height of the bounding box, respectively.

Next, Anchor Boxes are a set of predefined bounding boxes. They are defined to detect the aspect ratio of specific objects based on object sizes in the training dataset. The anchor boxes are moved across the entire image and then the neural network predicts the probability such as IoU to filter only potential boxes.

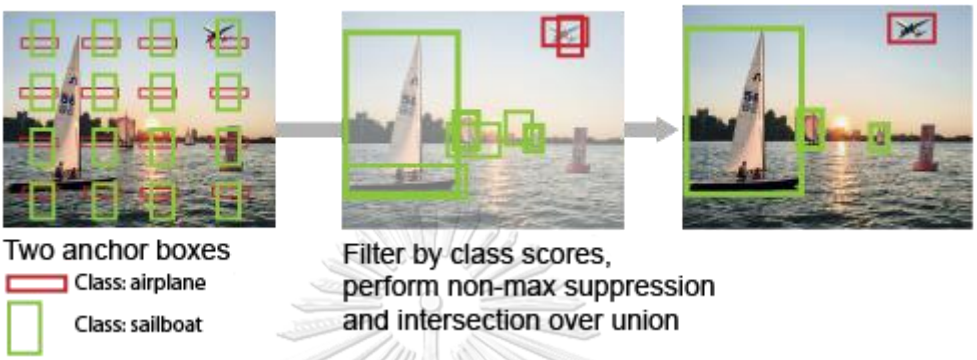


Fig. 2 Anchor boxes example

Intersection over Union (IoU) is a metric used to evaluate the accuracy of the predicted bounding box. It is a ratio of area of intersection to area of union between the prediction and ground-truth bounding boxes. The IoU value of more than 0.5 is considered a good prediction.

Fig. 3 IoU examples. Here the red bounding box represents ground-truth while the green bounding box represents prediction by the model.

Binary mask is a 2D array which has the same shape as the image. Each data point is either 1 or 0 (True or False) to define whether it belongs to the predicted instance.

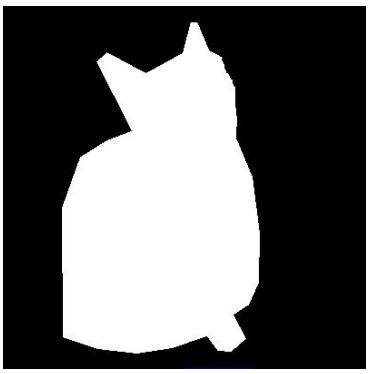


Fig. 4 Binary mask example

Lastly, Mean Average Precision, or mAP is an evaluation metric used to measure the accuracy of an object detector. The precision value equals to the number of true positives divided by the number of all positives. mAP is the average precision value over IoU of 0.5 to 0.95 with a step size of 0.05 and it is expressed as a percentage value.

2.1.1 Region-based CNNs

Region-based convolutional neural networks or regions with CNN features (R-CNNs) are pioneering approaches that apply deep learning models to object detection. Fast R-CNN, Faster R-CNN [4], and Mask R-CNN are some of the models that are developed as part of the improvement to the original R-CNN model [6].

R-CNNs

First, this model selects multiple proposed regions from an input image and then label their categories and bounding boxes. After that, it performs CNN forward computation to extract features from each proposed region and then predicts their categories and bounding boxes. The architecture of R-CNN is shown in Fig. 5.

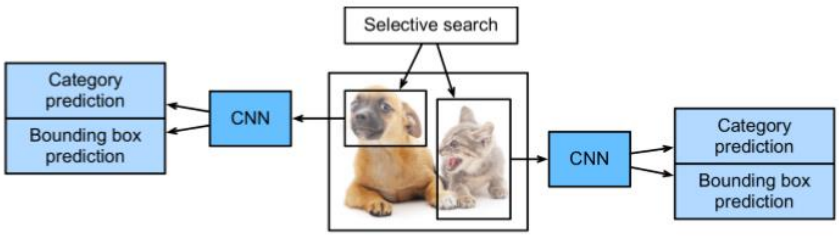


Fig. 5 R-CNN model

This model effectively uses pre-trained CNNs to extract image features, but its main drawback is the slow speed because the number of forward computations depend on the number of proposed regions.

Fast R-CNN

Fast R-CNN only performs CNN forward computation on a whole image once, solving the performance issue associated with the original R-CNN model. It introduces regions of interests pooling (RoI pooling) to extract features of the same shapes and a fully connected layer is needed to transform the output to a specific shape.

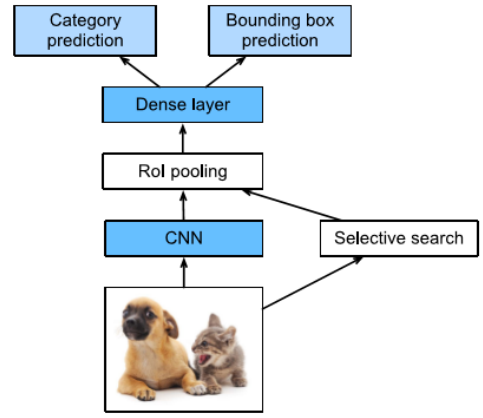


Fig. 6 Fast R-CNN model

Faster R-CNN

The selective search in Fast R-CNN generally generates many proposed regions. Faster R-CNN replaces selective search with a region proposal network to reduce the number of proposed regions generated and ensure accurate object detection at the same time. The other parts of the model are unchanged.

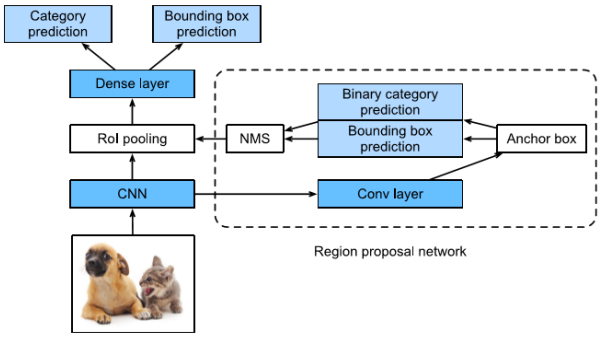


Fig. 7 Faster R-CNN model

Mask R-CNN

Mask R-CNN replaces RoI pooling layer in Faster R-CNN with RoI alignment layer to retain spatial information on feature map. This makes Mask R-CNN more suitable for pixel-level predictions. Then, it uses an additional fully convolutional network to predict pixel-level positions of objects.

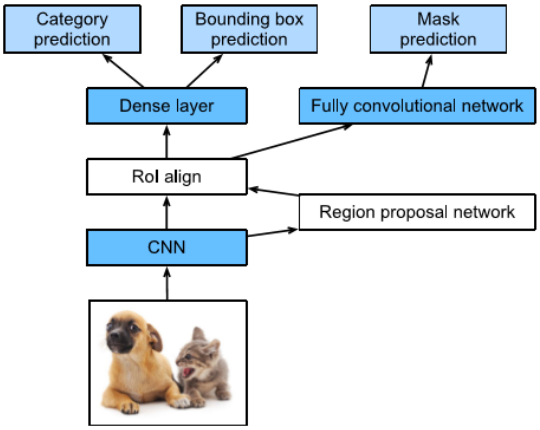
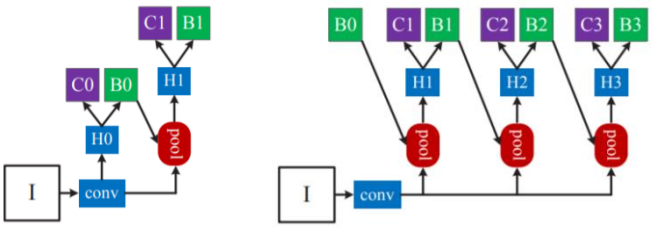


Fig. 8 Mask R-CNN model

2.1.2 Cascade R-CNN

Cascade R-CNN [2] is a multi-state object detection architecture developed from Faster R-CNN which is explained in the previous section. This model is proposed to solve 2 main problems: overfitting in training and inference-time mismatch between the IoUs for which the detector is optimal and those of the input hypotheses. These two main factors largely contribute to a performance degradation during detection.

This model decomposes difficult bounding box regression task into a sequence of simpler processes. The architectures are shown in Fig. 9.



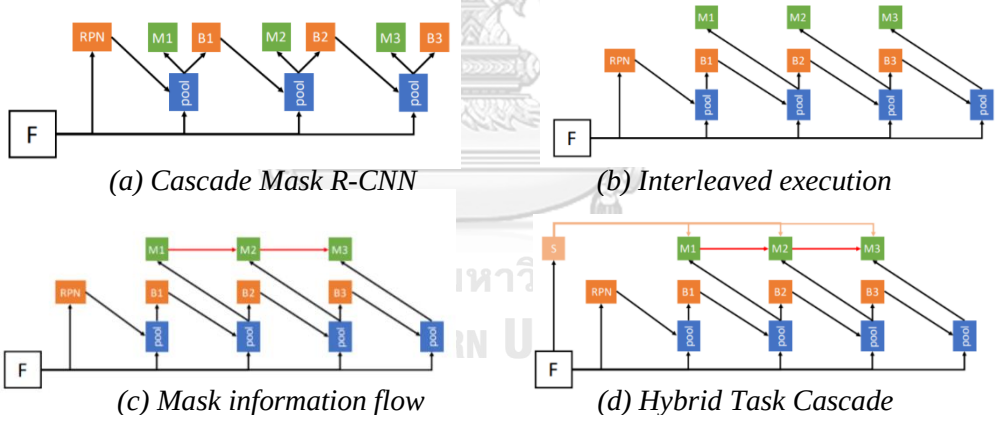
(a) Faster R-CNN *(b) Cascade R-CNN*

Fig. 9 The architecture of Cascade R-CNN comparing with Faster R-CNN

"I" is input image, "conv" is backbone convolutions, "pool" is region-wise feature extraction, "H" is network head, "B" is bounding box, and "C" is classification. "B0" is proposal in all architectures.

2.1.3 Hybrid Task Cascade

One of the most successful instance segmentation models, Cascade Mask R-CNN is the combination of the previous explained models: Cascade R-CNN and Mask R-CNN. It exploits the advantages of cascade architectures and to achieve better result. Hybrid Task Cascade (HTC) [8] improves on this Cascade Mask R-CNN by fully leveraging the reciprocal relationship between detection and segmentation. It interweaves these two branches to form a multi-state processing instead of performing them separately. Moreover, it creates a direct path to reinforce the information flow between mask branches and adds an additional semantic segmentation branch, which can ensure better background distinguishing abilities. The development of HTC from Cascade Mask R-CNN is shown in Fig. 10.



(a) Cascade Mask R-CNN *(b) Interleaved execution*

(c) Mask information flow *(d) Hybrid Task Cascade*

Fig. 10 The evolution from Cascade Mask R-CNN to Hybrid Task Cascade

2.2 Components of Deep Neural Network

2.2.1 Region Proposal Networks (RPN)

The RPN consists of a fully convolution network that shares convolutional features with the detection network. An image of any size is taken as an input of a Region Proposal Network (RPN). The outputs from this network are a set of bounding boxes with a score of objectness for each box. The network is trained end-to-end to generate many different region proposals with high accuracy.

The selective search in Fast R-CNN is replaced by this network and the results show that it can reduce computational costs, while still keeping the same effectiveness in terms of accuracy. The Faster R-CNN architecture explained in previous section is developed by merging the Fast R-CNN with RPN.

2.2.2 Residual Network (ResNet)

In deep convolutional neural networks, multiple levels of extracted features can be satisfied by adding more layers to the networks. As the number of layers in a deep neural network increases, the difficulty in training also increases due to higher training error.

One way to solve this problem is to implement residual functions with reference to the input layers. A residual learning framework is proposed to enable deeper neural network training. Previous studies [9] have shown that neural network depth can be as deep as 152 layers in ImageNet [10], or 8 times deeper than VGG nets [11]. Other analysis on CIFAR-10 dataset also shows network depth with 100 and 1000 layers [12].

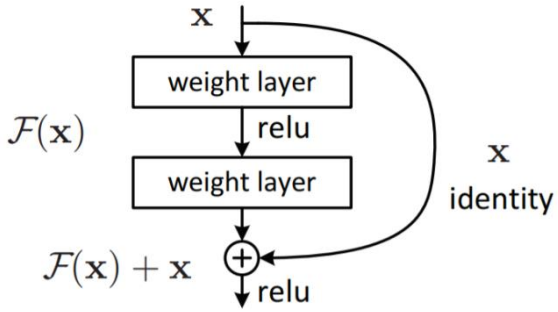


Fig. 11 A block of residual learning

2.2.3 Fully Convolutional Networks (FCN)

In image classification, an image goes through convolutional layers followed by fully connected layers. During that process, the image is downsized and is output as one predicted label. On the other hand, in semantic segmentation, the fully connected layers before the end of the CNN must be adjusted to convolutional layers. Therefore, it is called Fully Convolutional Networks (FCN) [13, 14]. After the convolutional layers, the output must be upsampling via deconvolution because the output size is scaled down in CNN. Then, the features extracted from different levels in CNN are combined to create a semantic segmentation result.

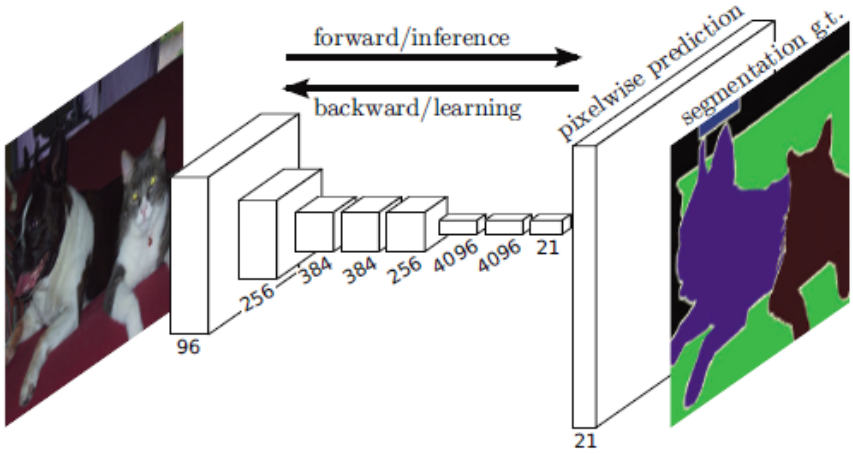


Fig. 12 Fully convolutional networks

2.3 3D Computer Vision

2.3.1 Depth Camera Technologies

Standard digital cameras provide us with output that is represented by a 2D grid of pixels, where each pixel contains several numeral values. For a common RGB color space, each pixel will have information pertaining to Red, Green, and Blue channel separately. Usually, an 8-bit integer is used to represent data for each channel, and hence, the integer value can vary from 0-255 in an (R, G, B) format. As an example, for a completely white pixel, the values must be (255, 255, 255), or a fully bright green pixel must be (0, 255, 0). The mixing between the different values of each color channel will give rise to the different colors that we can observe in photographs. In contrast, a depth camera has pixel which contains information representing the spatial distance (or "depth") of that pixel to the camera itself. In some depth cameras, a pixel may contain both the RGB and depth information - these are often referred to as RGB-D cameras, where each pixel contains 4 values which are Red, Green, Blue, and Depth values.

When it comes to calculating depth, there are several technologies being used by today's depth camera such as structured light and coded light, stereo depth, and time-of-flight and LiDAR, just to name a few. The working principle for each technology will be discussed below, as well as their advantages and shortcomings.

Structured Light and Coded Light

Structured light and coded light depth cameras rely on projecting a specific pattern of light, such as a series of stripes or dots, in an infrared range [15] onto a scene. If the scene contains some sort of a topology or a three-dimensional shaped surface, pattern light that was projected will appear to have a deformed pattern. The distance from the camera to the scene can be calculated based on the discrepancy between the actual image and the expected image of the patterned light. Unfortunately, these types of camera are sensitive to noises in the environment due to inference from other cameras or devices that also emit infrared light. Therefore, they tend to work best indoors and over a short range. Such technology is often used in gesture recognition and background segmentation. Some of the new cellphone camera relies on this technique to authenticate user when unlocking the phone using facial recognition [16].

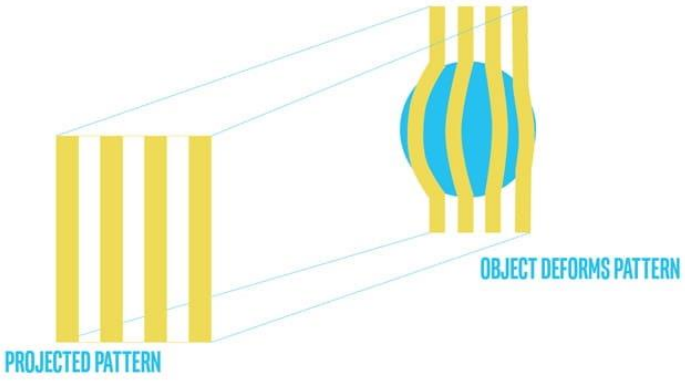


Fig. 13 Demonstration of structured light and coded light depth cameras. Note how the patterned light is deformed when 3D object is presence in a scene

Stereo Depth

Stereo depth cameras rely on two sensors and a small space between them. Given the known distance between the two sensors, the two images gathered from each sensor can be compared to estimate the distance of the target object from the camera. Interestingly, this technique is similar to how human eyes perceive depth from our two eyes, or how astronomers measure the distance of a star. Since the stereo technique does not rely on projecting a patterned light, this kind of camera can work well in most lightning conditions, including indoor and outdoor usage. Moreover, there are no

interference with other cameras, which opens up the possibility of using multiple cameras at same time.

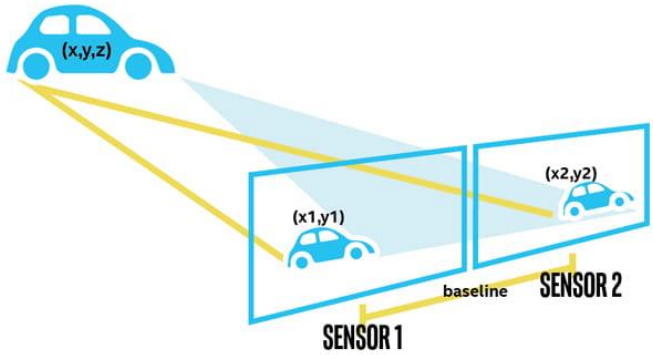


Fig. 14 Demonstration of stereo depth cameras. Images gather from sensor 1 and sensor 2 are used to calculate depth information

Time-of-Flight (ToF) and LiDAR (Light Detecting And Ranging)

A typical ToF or LiDAR emits a light of a certain wavelength onto a scene. Then, it detects how long does it take for that light to bounce off the different objects and be reflected to the sensor. Since the speed of light is known, the sensor can calculate distance traveled between the camera and the object. LiDAR offers some of the best resolution for measuring depth, with accuracy of up to millimeter in some cases [17]. It has become one of the main sensors used for 3D environment mapping and ADAS (Advanced driver-assistance systems) [18]. Like coded light and structured light cameras, time of flight cameras are vulnerable to noises in the surroundings. For examples, the sensors might be affected by the light traveling from another camera or sunlight.

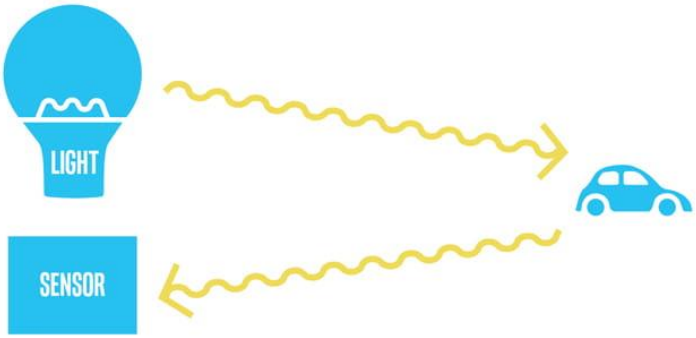
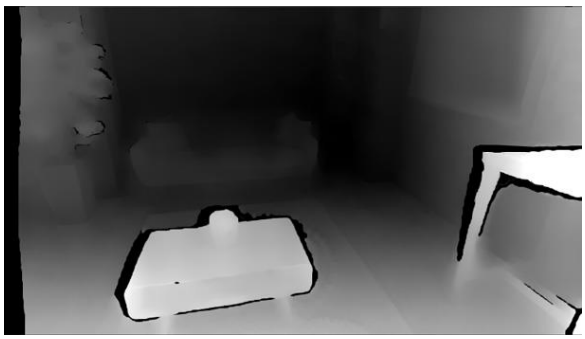


Fig. 15 Demonstration of time of flight cameras

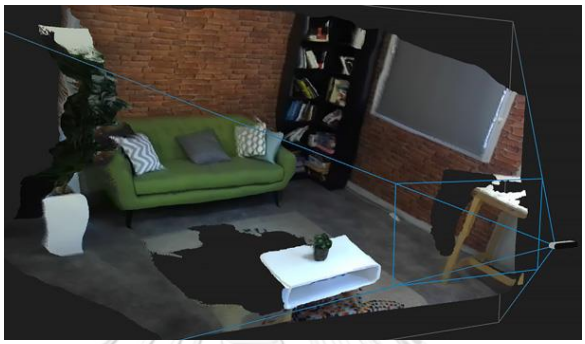
2.3.2 Depth Image vs 3D Point Cloud

Depth image and 3D point cloud can be obtained from a depth sensor or a depth camera. These two data representations contain the same information but are not identical. In a depth image, each (x, y) pixel represents a distance measured from an object in the scene to the camera. This measurement is often referred to as the depth value (z), which has a unit of length, whereas the x and y values are in pixel unit. On the other hand, a 3D point cloud is a set of (x, y, z) points in space, where the X, Y, Z coordinates all represent unit of length that measure the distance from those individual points to the camera.

Depth image can be visualized in a 2D image by using a heatmap with false color, or an 8-bit greyscale, corresponding to a particular depth value, as it represents a certain viewpoint of a 3D scene. The visualization for a 3D point cloud is different, however. The 3D point cloud can be visualized as a depth map in three-dimensional space that represents the external surface of the scene. In addition, the point cloud can display the true color value of a particular point in a scene, while the depth image cannot. It is worth noting that a depth image can also store color values, but it is not able to represent both the color data and depth data at the same time in a 2D image. Fig. 16 shows the visualization of these two types of data representation of the same scene.



(a) Depth map



(b) 3D Point cloud

Fig. 16 Visualization of a Depth map and a 3D point cloud

2.3.3 Centroid of a 3D point cloud

There are several different ways to compute the center of a centroid of a 3D point cloud cluster. The most common techniques are central feature, mean center, and median center. The central feature selects the center point to be the point which has the shortest accumulative distance to all the other points belonging to the point cloud. This can be found by iterating through all the points in the cluster and calculating the sum of distances of each point to all the other points and selecting the point with the least sum. The mean center technique is quite simple - it can be found simply by calculating average values of each x, y, and z components belonging to the cluster. The median center technique is similar to the mean center, but it finds the median of each coordinate components instead of finding the average values. The mean and median center techniques are different from the central feature in that the calculated centroid of these two techniques may or may not correspond to an actual point in the point cloud cluster. For a 3D point cloud which only has texture data of an external surface of an object or a hollow object, the centroid should be calculated using the latter two techniques since the calculation using the first method would yield an actual point on the surface, which is obviously not the correct centroid for these types of object.

2.4 Framework

2.4.1 Deep Learning Frameworks

At this time, there are many existing deep learning frameworks such as TensorFlow, PyTorch, Keras, MXNet, Microsoft CNTK, Deeplearning4j, and Caffe. Most of them are open-source. Each framework has its own specific applications, advantages, and disadvantages. In this section, we will give a brief overview of some of the most popular and widely used deep learning frameworks. The comparison between them is summarized in Table 2.

TensorFlow

TensorFlow is the most famous deep learning library written in Python and C++ developed by Google. It uses dataflow graphs, which are structures that define how data flow through a series of processing nodes. Each node represents a mathematical operation, and the connection between nodes is a multi-dimensional array called Tensor. Moreover, it offers TensorBoard [19] for data monitoring and visualization. This can be used to track the loss and accuracy of the model being considered. Many companies - like Uber, Airbnb, and Twitter - have employed TensorFlow in their platforms [20].

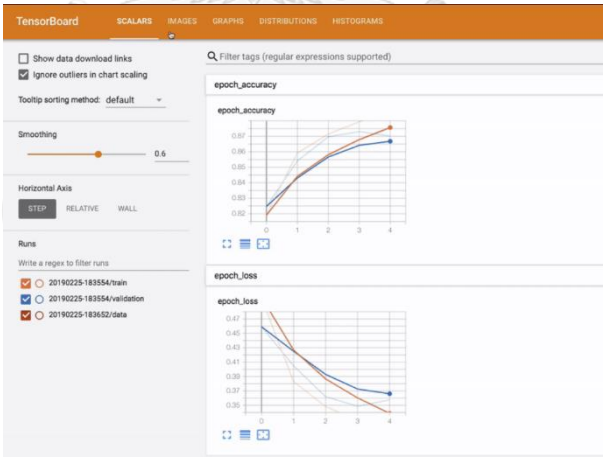


Fig. 17 Example of TensorBoard

PyTorch

PyTorch is a machine learning framework based on Torch which is an open-source package, but Python language is used instead of Lua [21, 22]. The data structure is also a tensor, which is very similar to NumPy arrays, yet it can be accelerated with GPU. In addition, this framework allows us to define our graph dynamically unlike other packages. PyTorch is flexible and fast, so it is suitable for deep learning research. It is developed and used by the social media giant, Facebook. Moreover, this framework is used in research by Oxford and IBM, and it can also work effectively with cloud platform like Amazon Web Services (AWS).

Microsoft CNTK

Microsoft CNTK, or Microsoft Cognitive Toolkit, is developed by Microsoft. It is used in popular Microsoft products such as Xbox, Cortana, Skype, and Windows Operating System. The CNTK provides neural networks in the form of directed graphs by using a series of computational steps. It also supports various programming languages such as C#, C++, Python, and Java.

Table 2 : Comparison of popular deep learning frameworks

	TensorFlow	**PyTorch**	**CNTK**
Developer	Google	Facebook	Microsoft
Supported languages	C++, Python, Java, JavaScript, Go, Swift	C++, Python	C++, Python, C#, Java, .NET
Main advantages	It offers Tensorboard for data monitoring and visualization and it has large community support.	It has various pre-trained models, powerful debugger tools, and a user-friendly design. It also supports distributed training.	It is easy to integrate in most enterprises and has reliable performance.
Main drawbacks	It is relatively slow comparing to other frameworks and also difficult to debug.	It has less community support and does not have visualization tools.	It has less community support.

2.4.2 MMDetection

Open MMLab Detection or MMDetection [5] is an open-source object detection toolbox and benchmark based on PyTorch [21]. It is developed by Multimedia Laboratory, CUHK. This framework has modular design and up to date, so it can be effectively used in object detection researches. A great number of architectures are implemented in this framework including the state-of-the-art models. Moreover, it has been approved to be highly efficient comparing to other popular frameworks.

2.4.3 Robot Operating System (ROS)

In a robot system, a lot of software tools are needed to control, drive, and perform computer vision tasks. A Robot Operating System, or ROS, attempts to gather all these tools together into a unified framework. The main goal of ROS is to support code reuse in robotics research and development [23]. The processes or computations in ROS are executed individually in each node. A node represents a set of runtime processes performing computation. These nodes can be combined to form a package, which is the main unit for organizing software in ROS. It is proven to be a very suitable tool in many robotic researches.

ROS has three levels of concepts: [24] Filesystem level, Computation Graph level, and Community level. These three levels are briefly explained below.

First, a Filesystem level concept involves ROS materials we meet on disk including packages which are the main unit for managing software in ROS. ROS runtime processes, ROS-dependent library, datasets, or anything else may be contained in a package hence they can be effectively managed together. Other than the packages, Message types (which define data structure of messages) and Service types (which define the request and respond data structures) are also parts of a Filesystem level concept.

Next, the Computation Graph level covers the peer-to-peer network in ROS processes which transfer data between one another. The basic computation graph concepts are ROS nodes, master, messages, services, topics, bags, and anything else that provides data to the network. Each ROS node can be viewed as a process that is tasked with performing a specific type of function. For examples, one node might control the motors of a robot, and another node might localize the robot, while another

node might perform path planning. ROS master enables the communication of nodes in a computation graph by providing name registration. It also contains the parameter server which allows data to be stored by key in a central location. ROS topic acts like a strongly typed message bus, allowing nodes to send or receive certain types of ROS messages through publishing and subscribing to the topic. A single topic allows multiple publishers or subscribers to access concurrently. ROS nodes connect to the other nodes directly while the ROS master only gives lookup information (similar to DNS server). The subscribers will request connections to the publishers and connection will be initiated over an agreed protocol. The most common protocol used in ROS is TCPROS, which uses standard TCP/IP sockets. Another important concept in this level is ROS bags. These are mechanism for saving and playing back ROS message data such as those outputted from a sensor, which can be retrieved and visualized at a later time.

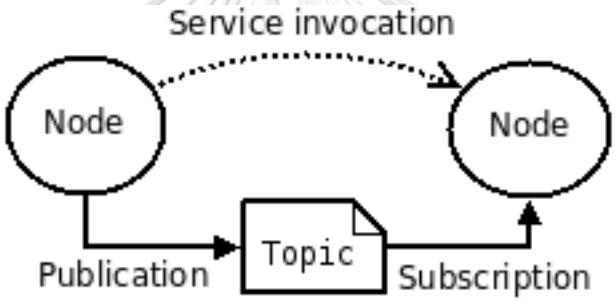


Fig. 18 Demonstration of ROS nodes communication

Finally, the ROS Community level concept is ROS materials that allow users and developers to exchange software and knowledge. These concepts include distributions, repositories, the ROS wiki, etc.

In addition to the three levels of concept, ROS also states 2 types of names, which are Graph Resource Name and Package Resource Name. The first type of names provides hierarchical naming structures, which is used in the ROS Computation Graph including nodes, topics, services, and parameters. This kind of naming structure is useful, especially when ROS system grows larger and becomes more complicated. The other type of names, Package Resource Name, is used in the ROS Filesystem level for referring to files and data types on a disk with an abbreviated notation. It can simply be constructed by listing the name of a package, followed by a name of a resource that you want to access. For examples, a package resource name of “std_msgs/PointCloud2” is a shorthand version for “absolute/path/to/std_msgs/PointCloud2.msg”.

CHAPTER 3 METHODOLOGY

3.1 Hardware

The robot system used in this experiment is a 7-Axis articulated arm robot system from Yaskawa Electric Corporation, Japan, with a model name Motoman SIA5F [25]. It supports many applications such as assembling, machine tending, material handling, part transferring, and picking-and-placing with a payload of up to 5.0 kg. It has a horizontal reach of 559 mm and a vertical reach of 1007 mm. The controller model is FS100. The robotic arm is connected to and controlled by several computers used for performing different tasks such as path planning and gripper controlling. The entire robot system is operated and communicated on Robot Operating System (ROS) network [23]. Several depth cameras are set up in the system to perform workspace calibration and computer vision. One of the cameras fixed near the end-effector is a stereo depth camera. It is used for computer vision to enhance object manipulation task like pick-and-place with a robotic arm. The model of the depth camera used is Intel RealSense D435i [26]. It is connected via a USB to a desktop computer running high performance GPU, Nvidia GeForce RTX 2080Ti [27], which is suited for performing high computation tasks like deep learning. The data flow for the image processing of our system is shown in Fig.19. Computer A directly receives the raw data from the depth camera which include RGB and depth data. The RGB data are used for instance segmentation and then combined with depth data to create 3D point cloud information. The processed data are sent to other computers in the network to be used in other processes like path planning and grasp posture optimization.

This work focuses on the data flow starting from the acquisition of RGB and depth data via depth camera to the processed data generated by Computer A, as depicted in Fig. 19.

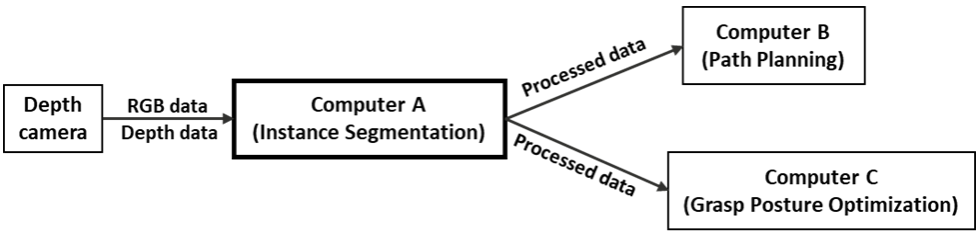


Fig. 19 Data flow example of image processing part in a robot system

Table 3: Specifications of the desktop computer used in the robot system

CPU	Intel Core i9-9900K
RAM	DDR4 32 GB
GPU	Nvidia GeForce RTX 2080Ti

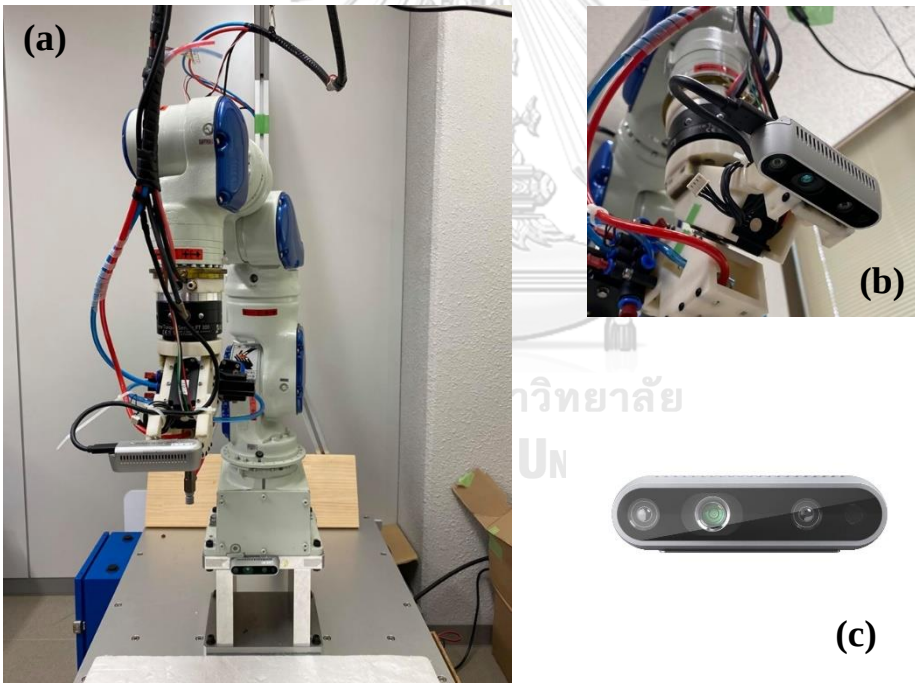


Fig. 20 Manipulator and the camera used for computer vision: (a) Motoman SIA5F Manipulator, (b) Depth camera mounted at the end-effector, (c) Intel RealSense D435i Depth camera

3.2 Deep Learning Implementation

In order to achieve a detector which is able to classify and localize food objects, we use deep learning as a method to develop instance segmentation model. The model starts with the initial weights and goes through the training and evaluation to obtain

better weights according to the training dataset. Finally, we will be able to classify different food objects from images, which is one of our main objectives.

3.2.1 MMDetection Toolbox

In this experiment, we use OpenMMLab Detection Toolbox and Benchmark, or mmdetection [5] as a framework for performing deep learning and instance segmentation. This framework provides tools and python APIs for model training, evaluating, and testing and is open-source. Moreover, it offers a lot of competitive models to use including backbones, methods, and the state-of-the-art models. MMDetection uses distributed training which splits the training workload and shares them among multiple processors.

This toolbox implements a contemporary and popular deep learning framework, PyTorch [21] as its backend which provides various deep learning APIs and also offers a lot of pre-trained models and powerful debugger tools.

This framework can be run either locally on a desktop computer with a GPU, or on a cloud-based service such as Google Colaboratory (Colab) [28]. In this study, we perform all operations of this framework using both local computer and Google Colab platform. For the local computer use case, we need to install all the necessary dependencies needed for MMDetection such as Torch, Mmcv, and Numpy on the disk. All of the APIs and tools are ready to be used once you have installed the framework correctly. For Google Colab use, a python notebook needed to be created in order to use the framework's functionalities. The Colab provides basic dependencies for deep learning such as Torch and Matplotlib, so we do not need to install them, but MMDetection is still needed to be setup every time before using the notebook.

3.2.2 Deep Learning Architectures

We select two deep learning architectures designed for instance segmentation tasks to train detector models in this study, namely, Cascade Mask R-CNN and Hybrid Task Cascade [8] since both of them are in the top-ranking for instance segmentation on COCO test-dev benchmark [29]. These two models are quite similar because they are based on cascade architectures. The second model, HTC, was developed based off Cascade Mask R-CNN, which is the combination of Cascade R-CNN [2] and Mask R-CNN [3]. The development of HTC from Cascade Mask R-CNN is shown in Fig. 10. Evaluating using COCO average precision, the HTC achieves the box AP of 43.2% and

the mask AP of 38.0%, which exceeds that of Cascade Mask R-CNN having the box AP and the mask AP of 42.5% and 36.5%, respectively.

Cascade R-CNN has multiple stages of bounding box regressions where the output bounding boxes from a previous stage are taken as new region proposals and inputs of a current stage. However, the 3-stage Cascade R-CNN is commonly used because researchers found that increasing more than 3 stages does not further enhance the effectiveness. Cascade Mask R-CNN is developed from Cascade R-CNN by adding segmentation branches in parallel to the bounding box regression and classification [30]. In this case, the mask branches in each cascade stage do not affect one another as shown in Fig. 10a.

HTC, on the other hand, improves Cascade Mask R-CNN by eliminating this problem. It uses interleaved execution for bounding box branches and mask branches, then adds a direct mask information flow to connect between mask branches. Finally, it includes semantic segmentation branch to the model. This last step differentiates HTC from Cascade Mask R-CNN because this now requires extra training due to the supplement. In this study, we use the HTC architecture without the semantic segmentation branch, so we do not need to acquire extra training data.

3.3 Model Training

In the MMDetection toolbox, a config file contains a structure of an architecture and also other values such as location of the dataset, number of training epochs, and many important parameters. We select the methods or the architectures for training by specifying a path of a config file. Before training, we need to make sure that we have specified all the desired parameters. Once the dataset is prepared, we can perform training using the train script provided in the toolbox.

In this experiment, we perform model training with the same dataset for both Cascade Mask R-CNN and HTC architectures on both local PC and Colab. The selected training parameters are the same values as shown in Table 4.

Table 4: Training parameters

Training Parameter	Value
Backbone	ResNet-50
Style	PyTorch
Learning Rate Scheduler	1x

For the local training, a single high-performance GPU, Nvidia GeForce RTX 2080Ti is used and the models are trained for 1000 epochs. For Colab training, the GPU used is Nvidia Tesla T4 provided by Google Colab and is trained for 300 epochs for each architecture.

3.4 Datasets

The deep learning is conducted on our own images dataset which is annotated in the same style as COCO dataset [31]. The dataset consists of 807 images, of which 765 images are annotated, while the rest are images of the laboratory environment. There are 13 categories in the dataset: Japanese lunch box (Bento Box), bologna sausage, potato chip, Japanese fried chicken (Karaage), Japanese rice ball (Onigiri), and 8 different kinds of sushi which are shrimp (Ebi), squid (Ika), red caviar (Ikura), salmon, Japanese omelette (Tamago), tuna, eel (Unagi), and sea urchin roe (Uni). These categories are chosen based on food commonly found in Japanese supermarkets.

The annotations are done via a software called COCO-Annotator [32] where all the images are annotated by hand. First, we input sets of images to a certain directory and then the application locates and displays all the images in that directory on the GUI of the software. The categories must be defined in the program before we can label using a polygon tool image by image. After that, the COCO annotation file can be exported in JSON format.

We split this dataset into train set and validation set, consisting of 720 images and 87 images, respectively. The average width and height of the images are 695 pixels by 527 pixels. The detail of the dataset is summarized in Table 5. Examples of annotated images of the food objects belonging to the different categories are shown in Fig. 21.

Table 5: Dataset detail

Category	Images		Annotations	
	Train	Val	Train	Val
Bento box	142	17	142	17
Bologna	27	3	195	23
Potato chip	55	9	82	11
Ebi nigiri	105	11	139	17
Ika nigiri	101	10	138	13
Ikura nigiri	49	6	84	9
Karaage	104	16	474	59
Onigiri	96	10	325	27
Salmon nigiri	103	13	197	21
Tamago nigiri	98	10	159	19
Tuna nigiri	53	4	117	7
Unagi nigiri	49	5	118	11
Uni nigiri	49	6	104	12
Surroundings	38	4	-	-

Fig. 21 Examples of annotated images

3.5 Point Cloud Extraction

3.5.1 Via Depth Topic

The depth camera streams data which consist of RGB and depth frames that can be used to generate point cloud ROS message. We use realsense2_camera package to access data from the camera. The package's node, called /camera/realsense2_camera, publishes raw camera data which are RGB, depth, and camera information to different ROS topics. The /pointcloud_masking node subscribes to these topics and accepts the data as inputs which will then be transformed into point cloud information of food objects. The subscriptions are done via callback functions. The RGB data are used to perform inference with the trained model, providing detection and segmentation information (mask) of the food objects being detected. These masks are binary masks in an array structure with the same dimension as the original RGB image, but the values in the array are stored in Boolean format as true or false instead. The true value indicates that the pixel is in a segmented object, and vice versa. This binary mask is then applied to the depth data from the subscribed depth topic. The depth data is an array consisting of depth value for each pixel and it has the same dimension as the binary mask array. This processing step is demonstrated with a 6x8 size image as shown in Fig.22, where Fig.22a represents an RGB frame, Fig.22b represents a mask array, Fig.22c represents a depth frame, and Fig.22d represents the resulting depth frame after applying the mask array.

Z11	Z12	Z13	Z14	Z15	Z16	Z17	Z18
Z21	Z22	Z23	Z24	Z25	Z26	Z27	Z28
Z31	Z32	Z33	Z34	Z35	Z36	Z37	Z38
Z41	Z42	Z43	Z44	Z45	Z46	Z47	Z48
Z51	Z52	Z53	Z54	Z55	Z56	Z57	Z58
Z61	Z62	Z63	Z64	Z65	Z66	Z67	Z68

(a) 6x8 RGB frame

0	0	0	0	0	0	0	0
0	0	0	Z24	Z25	0	0	0
0	0	Z33	Z34	Z35	Z36	0	0
0	0	Z43	0	Z45	Z46	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(b) 6x8 mask array

P11	P12	P13	P14	P15	P16	P17	P18
P21	P22	P23	P24	P25	P26	P27	P28
P31	P32	P33	P34	P35	P36	P37	P38
P41	P42	P43	P44	P45	P46	P47	P48
P51	P52	P53	P54	P55	P56	P57	P58
P61	P62	P63	P64	P65	P66	P67	P68

(c) 6x8 depth frame

F	F	F	F	F	F	F	F
F	F	F	T	T	F	F	F
F	F	T	T	T	T	F	F
F	F	T	F	T	T	F	F
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F

(d) The depth frame after applying mask

Fig. 22 The demonstration of applying mask to a depth frame

The depth values in the array are in millimeters and its actual dimensions are H x W x 1, where H and W are height and width of the image, respectively. After applying the mask to the depth frame, the values at the pixels which are not in the segmentation window will become zero. At this point, we have retained only the depth information of the desired area. The resulting depth frame is then transformed from a pixel coordinate into a metric coordinate in order to create 3D point cloud. To accomplish this, we need the camera intrinsic parameters which are used to map camera coordinates into image plane (world points). The camera intrinsic parameters are obtained from /camera/color/camera_info ROS topic via a callback function. The metric coordinate can be calculated according to Eq. (1).

$$\begin{aligned} z &= depth/1000 \\ x &= z(u - K_{13})/K_{11} \\ y &= z(v - K_{22})/K_{21} \end{aligned} \qquad (1)$$

, where x, y, z are the metric coordinates of a point, u and v are the horizontal and vertical pixel coordinates of a point, and K is the camera intrinsic matrix, respectively. Once we have obtained a set of 3D points in metric coordinates, we are able to create sensor_msgs/PointCloud2 to publish the information in the ROS system. In this case, the point cloud information of each category is published on different topics. For examples, /pointcloud_1 is a point cloud topic for category one, and /pointcloud_2 is a point cloud topic for category two. Furthermore, we can colorize the point cloud for each category to enhance visualization by adding a floating RGB value to each point in the set of 3D points, so that each point will now contain x, y, z, and an RGB value. The ROS diagram for this node is shown in Fig.23.

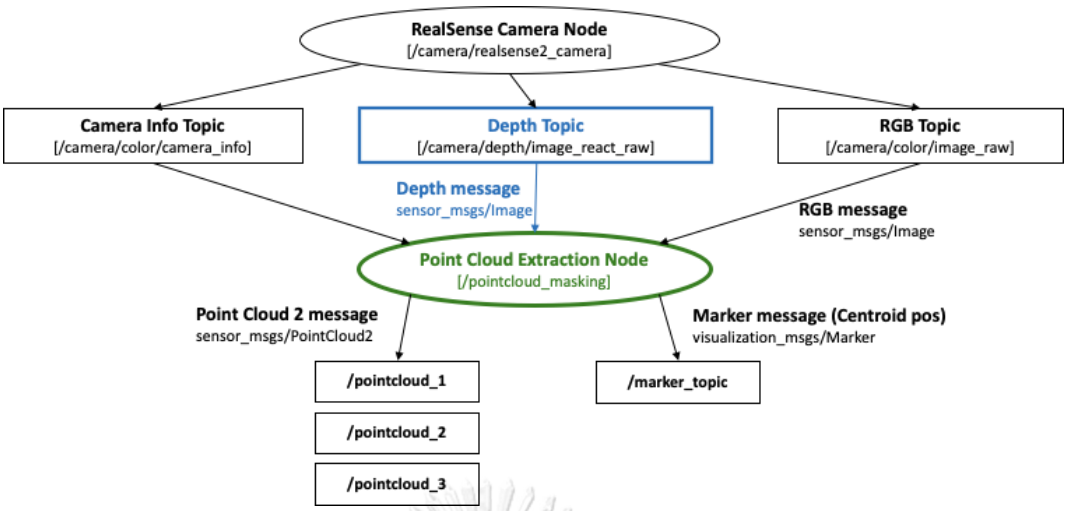


Fig. 23 Point cloud extraction in ROS via Depth Topic

3.5.2 Via Point Cloud Topic

We explained a method to obtain food object's point cloud information and publish that information in a ROS environment. There is also an alternative method that can be used to extract point cloud as outlined in Fig.24.

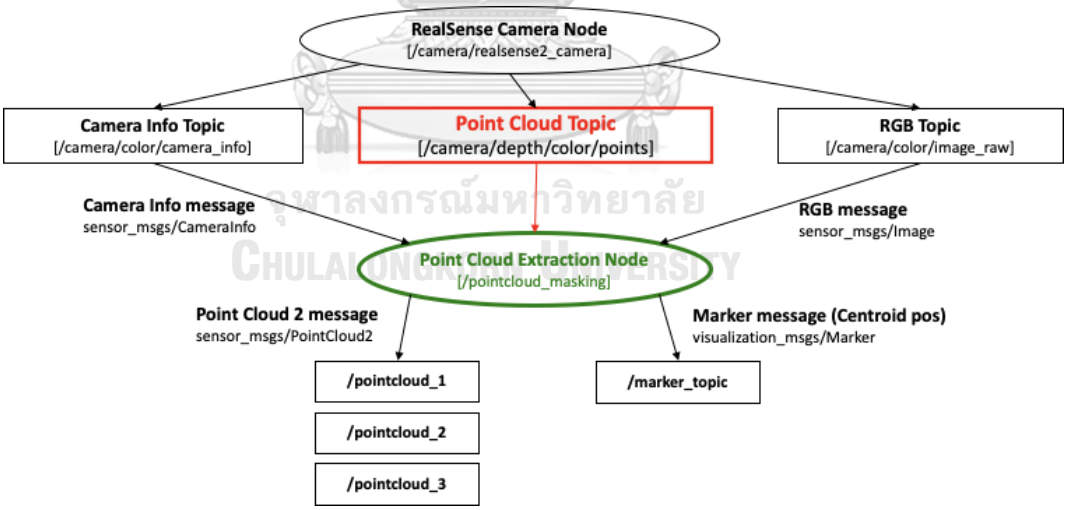


Fig. 24 Second method for point cloud extraction in ROS via Point Cloud topic

This second method differs from the first method in the way that it subscribes to the point cloud topic instead of the depth topic. One advantage of using this second method is that it simplifies the calculation in the /pointcloud_masking node. If the point cloud topic provides the depth registered point cloud message, we can easily obtain the

point cloud of the desired objects by applying the mask array to the point cloud array directly. The depth registered point cloud is an array with a dimension of H x W x 4. The number 4 implies each point has 4 attributes, which are x, y, z, and RGB values of a point. With this kind of point cloud information, we can apply a mask to it in a similar manner as previously shown in Fig.22. However, if the point cloud is not depth registered, we will need to perform an additional calculation. For example, after the inference step, we find all the possible contours (in a closed form) in the mask image and pick out the largest contour. Next, we generate a polygon of this contour, consisting of a set of vertices of the contour boundary. Then, we filter the desired points by determining only the points in the points array that reside in the polygon. Each point in the point cloud has its original RGB color from the camera's RGB data, hence we can see the object color as we see in a 2D RGB frame instead of a mono color as was seen when using the first method for point cloud extraction.

3.5.3 Centroid of Point Cloud

In the previous section, we discussed how individual point clouds are extracted from a depth camera data. Using the extracted point cloud data, we can calculate the centroid position of a point cloud by simply taking the average values of each coordinates x, y, and z values belonging to the segmented object. Then, we publish the centroid position of the segmented object in a ROS topic using /visualization_msgs/Marker ROS message. This information is especially useful for food automation involving pick-and-place operation.

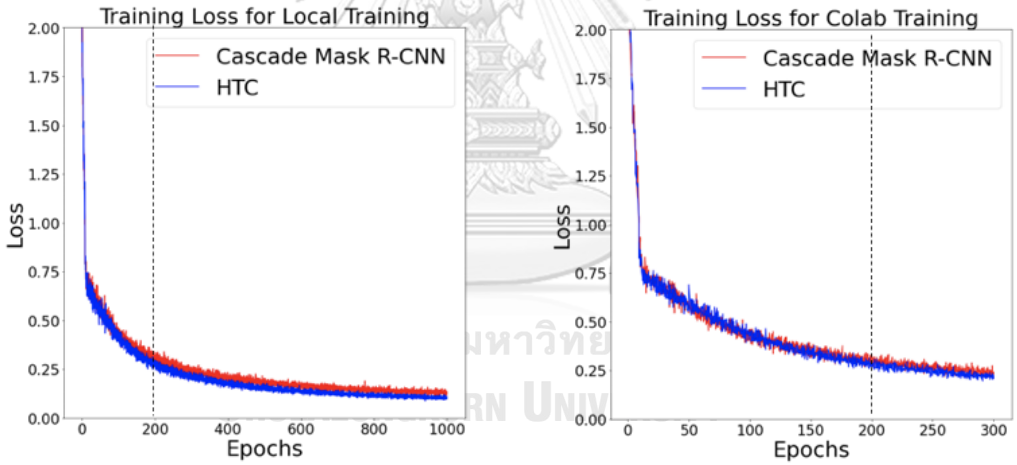
CHAPTER 4 EXPERIMENTAL RESULTS AND DISCUSSION

In this experiment, we separate the results into 2 main parts, namely, 2D segmentation and 3D point cloud extraction. For 2D segmentation part, we evaluate and compare between the trained Cascade Mask R-CNN and HTC models. We then test the trained HTC model under different scenario conditions to assess the accuracy and robustness of the model. For 3D point cloud extraction part, we show the point cloud extraction result in 3D visualization and compare the two methods used in the extraction process.

4.1 2D Segmentation

4.1.1 Model Evaluation

After training the models with 2 different instance segmentation architectures which are Cascade Mask R-CNN and HTC on both local pc and Google Colab cloud service, we plot the training loss curves comparing these 2 models as shown in Fig. 25.



(a) Local training *(b) Colab training*

Fig. 25 Training loss comparison

As you can see in Fig.25a and Fig.25b, the loss curves for HTC using local training and Google Colab, are slightly lower than that of Cascade Mask R-CNN after 200 epochs. Notice that the loss curve that was run on Google Colab only contains 300 epochs as compared to 1000 epochs when running on a local computer.

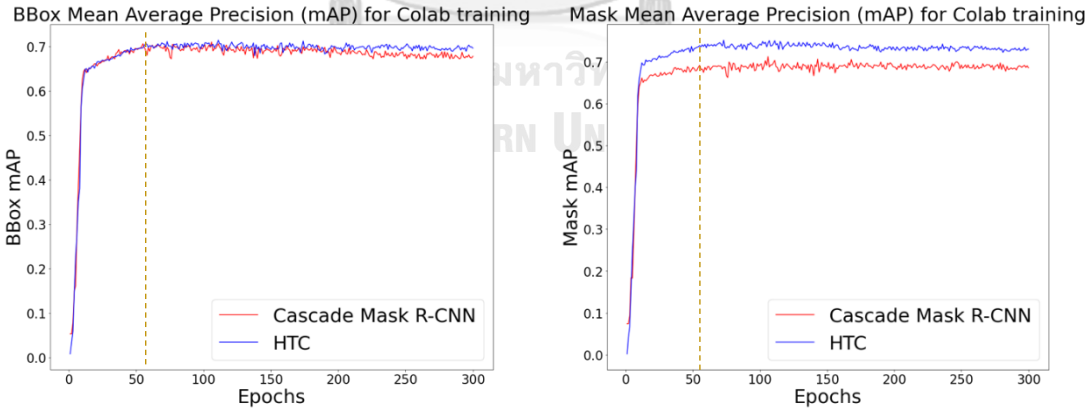
Next, we test the models using the test script provided by the toolbox. These models are evaluated using COCO average precision metric (mAP) that evaluate over IoU (Intersection over Union) of 0.5 to 0.95 with a step size of 0.05 and it is expressed

as a percentage value. The bounding box APs and the mask APs of each model are summarized in Table 6.

Table 6: Model Evaluation

Model	Local Training (1000 epochs)		Colab Training (300 epochs)	
	Box mAP (%)	Mask mAP (%)	Box mAP (%)	Mask mAP (%)
Cascade Mask R-CNN	66.9	68.7	67.8	68.5
HTC without semantic	67.4	73.8	69.6	73.0

The results show that the HTC models always have higher mean average precisions than the Cascade Mask R-CNN models in both bounding box and mask branches, and the results reveal the same characteristics for both local training and Colab training. Even though we train the models with a smaller number of epochs on Colab, the bounding box mAPs for both architectures are higher than those of local training. However, the mask mAPs for both architectures on local training are still higher. In this case, we might not have to train with such a large number of epochs because the performance of the models does not improve that much further. This will allow additional saving of computation power and time. The relationship between model evaluation and epochs for Colab training is shown in Fig.26.



(a) BBox mAP *(b) Mask mAP*

Fig. 26 Model evaluation for each epoch for Colab training

Fig.26a and Fig.26b show that mAP for both box and mask branches are saturated after approximately 50 epochs of training for both architectures. The box mAPs of these two architectures are approximately the same, but the box mAP of Cascade Mask R-CNN tends to slightly decrease as the number of epochs increases.

The results of mask mAP show that the HTC has better mask mAP than the Cascade Mask R-CNN. These results suggest that the number of training epochs for these two architectures on this dataset do not have to be as many as 300 or 1000 since the precisions of the models do not improve after 50 epochs. In this case, we could train the models for 100 epochs or less instead.

4.1.2 Segmentation Results of the test dataset

Examples of segmentation results are shown in the following figures. These results were obtained by testing images in the test set of the dataset using the test script. Both models are trained locally. The results show that the Cascade Mask R-CNN model has more overlapping bounding boxes and more false positive detections than the HTC model as shown in Fig. 27 where the yellow arrows show the occurrences of false positives. This is because the precision of the HTC is higher than the Cascade Mask R-CNN as we stated in 4.1.1. However, these two models show similar results in some segmentation images (Fig. 28).

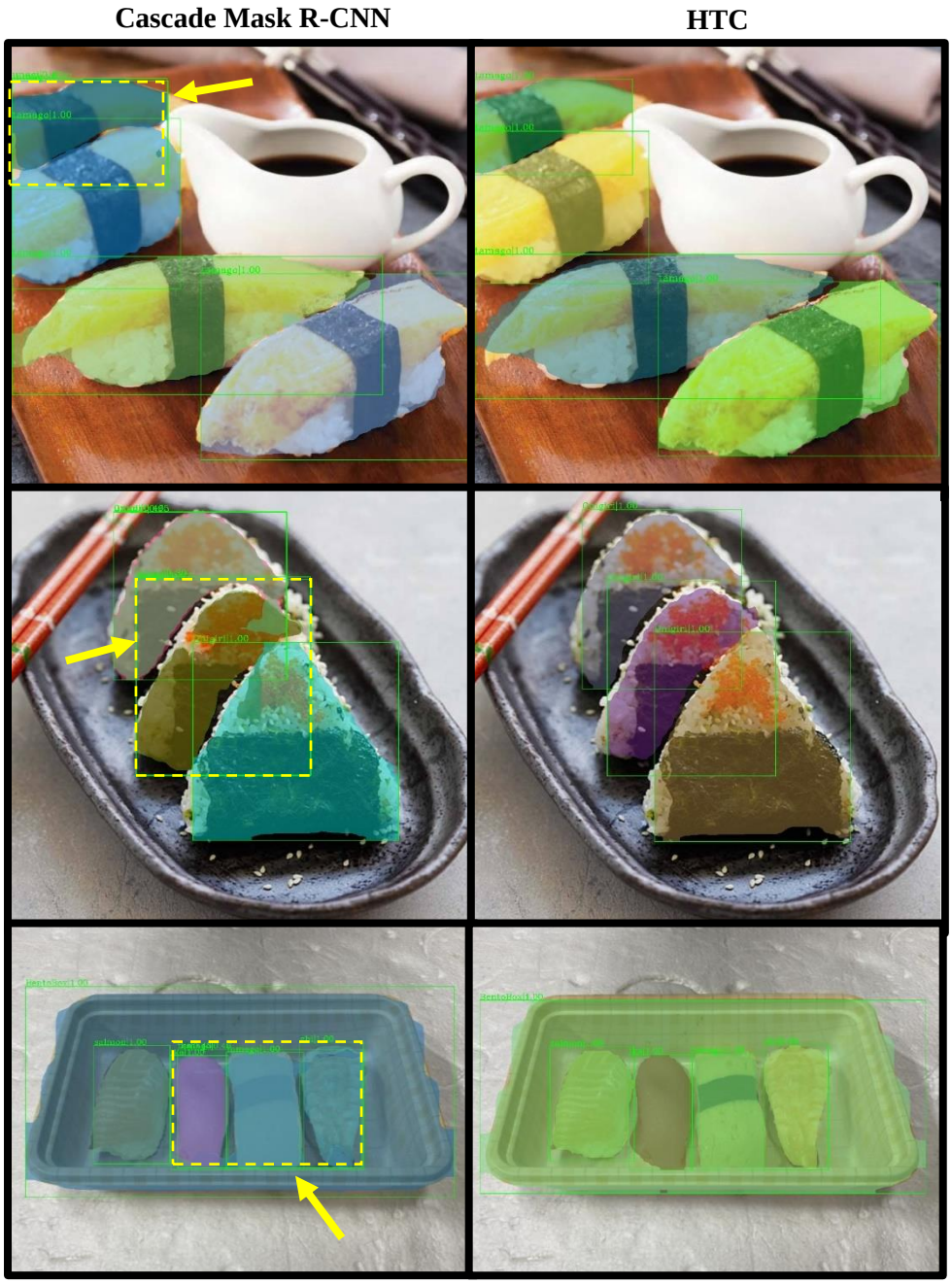


Fig. 27 Examples of segmentation results of Cascade Mask R-CNN (Left) and HTC (Right)

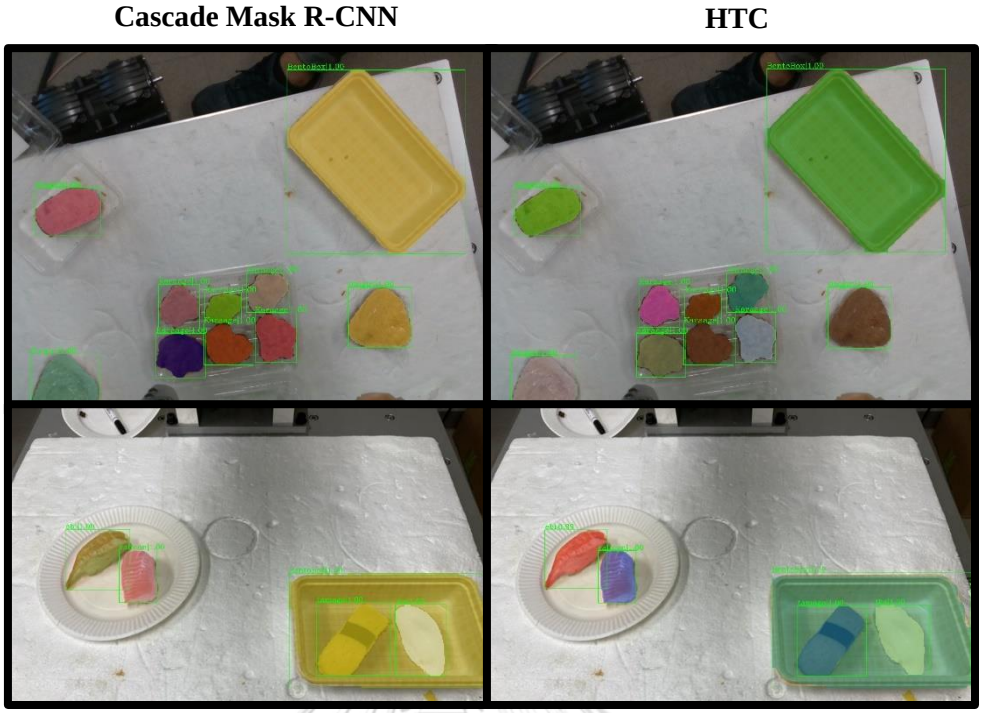


Fig. 28 Examples of segmentation results of Cascade Mask R-CNN (Left) and HTC (Right)

4.1.3 Scenario Impacts on HTC model

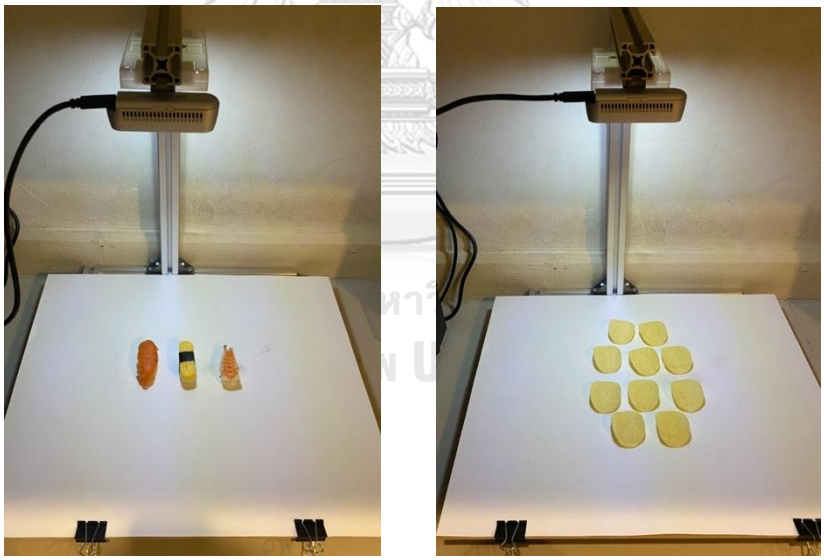
As part of the model evaluation, we also perform experiments to test how different scenario conditions might impact the result when using the HTC model. For this experiment, we construct a stationary frame for attaching Intel RealSense d435i camera to obtain a fixed field of view (FOV) throughout our test. We position the camera to be facing down from the top fixture to create a top-down view of the test condition. The base of the frame is made out of a thin wood plate that serves as a background for the scene. This background color can be changed simply by clamping different color papers to it. A brightness-adjustable LED light is also attached to the top fixture next to the camera, which allows for brightness adjustment in the experiment. The constructed stationary frame, camera attachment, and LED lighting are displayed in Fig. 29.

The food objects in our test are divided into 2 sets. The first set consists of 3 different types of sushi: salmon sushi, egg sushi, and boiled shrimp sushi – all of which are different in colors and each have its own detection classes. The second set consists of multiple pieces of potato chips, which have similar color and are belonged to the

same detection class. Examples of the experimental setup with two sets of food objects placed under test are shown in Fig.30.

(a) Framework *(b) RGB-D camera and LED light attachment*

Fig. 29 Experimental framework for scenario impacts on the HTC model



(a) Sushi *(b) Potato chips*

Fig. 30 Experiments of food objects on the test framework with white background color

The experiment consists of six scenarios to test the detection of HTC model under various conditions, including luminosity, background colors, addition of non-food objects, placements and positioning, incomplete food objects, and image resolutions. We capture multiple images of each scenario using both sets of food objects with the camera attached on the test setup. The image resolution used is 640 pixels by 480 pixels,

which is same resolution that we use to perform video streaming. The captured images are tested with the HTC model which is trained on Google Colab for 300 epochs. We perform inference on Colab with Nvidia Tesla T4 GPU using the developed scripts to automatically test the model, save output images, and log the prediction details of each image with score threshold is set to 0.5. Finally, the results are analyzed based on Precision and Recall values, which are briefly explained below.

I. Precision

A precision value is a ratio of true positives (TP) to the total number of predicted positives. It measures the accuracy of the model predictions and expresses how many percent of the prediction are correct. The precision is calculated according to Eq. (2).

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

II. Recall

A recall value or a true positive rate (TPR) is a ratio of true positives to the total number of ground truth positives. It measures the ability to find all of the ground truth positives of the model. It can be calculated as shown in Eq. (3).

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

, where TP is the number of true positives or number of predictions that are correct, FP is the number of false positives of number of predictions that are incorrect, and FN is the number of false negatives of number of ground truth objects that are not predicted [33]. These concepts can also be visualized as shown in Fig. 31.

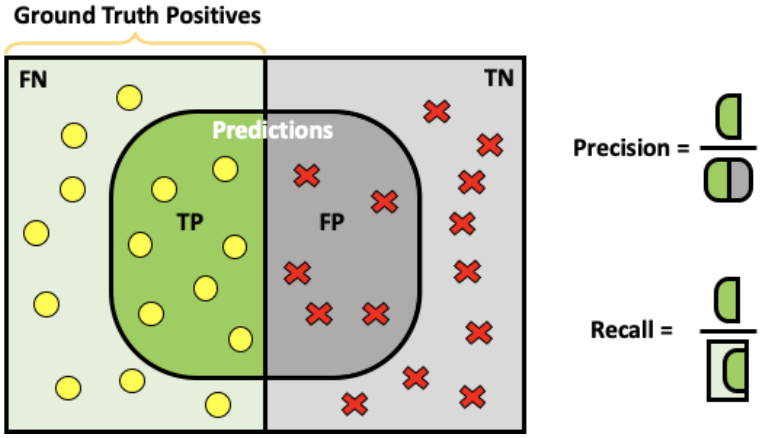


Fig. 31 Precision and Recall explanation

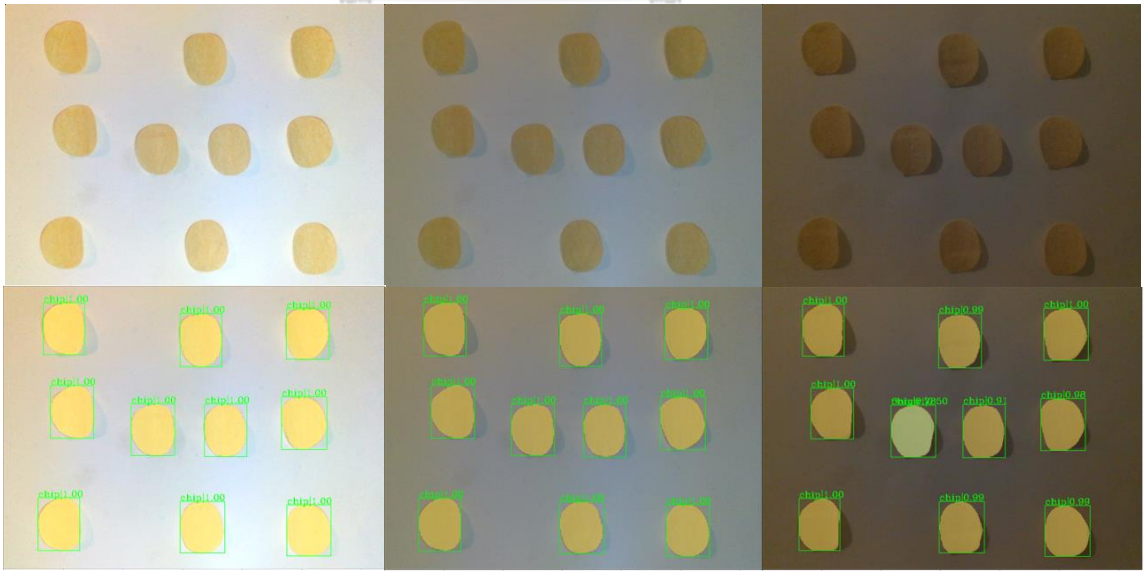
Having established the definition for the proposed metrics, we will use these metrics to evaluate the performance of our model under the different scenario conditions.

4.1.3.1 Luminosity

We postulate that the brightness of a scene captured by a camera may affect the performance of the prediction. Using the brightness-adjustable LED light attached on top of the framework, three different levels of brightness are defined:

1) Minimum level : LED turned off
2) Medium level : LED at half brightness
3) Maximum level : LED at full brightness

We perform this experiment on white background color and adjust the brightness using these 3 levels settings. 10 photos of 10 different scenes for each brightness level and each set of objects are performed. For the potato chips set, we randomly place 10 identical potato chips with random spacing between each one in a scene. Therefore, there are a total of 100 potato chips that are used in this experiment. For the sushi set, we use 3 different sushi of different types in a scene (total of 30 sushi). Some example images, along with their prediction results, are shown in Fig.32-33, Table 7, and Table 8 below.



(a) Maximum *(b) Medium* *(c) Minimum*

Fig. 32 Examples of potato chips detection results for 3 levels of brightness setting

(a) Maximum *(b) Medium* *(c) Minimum*

Fig. 33 Examples of sushi detection results for 3 levels of brightness setting

Table 7: Class predictions with IoU scores for 3 levels of brightness of potato chip set

Maximum	Medium	Minimum
chip,0.9999	chip,0.9999	onigiri,0.5004
chip,0.9999	chip,0.9999	chip,0.9995
chip,0.9999	chip,0.9999	chip,0.9990
chip,0.9999	chip,0.9999	chip,0.9982
chip,0.9999	chip,0.9999	chip,0.9981
chip,0.9999	chip,0.9999	chip,0.9917
chip,0.9999	chip,0.9999	chip,0.9890
chip,0.9999	chip,0.9999	chip,0.9873
chip,0.9999	chip,0.9999	chip,0.9795
chip,0.9999	chip,0.9999	chip,0.9094
		chip,0.7818

Table 8: Class predictions with IoU scores for 3 levels of brightness of the sushi set

Maximum	Medium	Minimum
ebi,0.9335 salmon,0.9997 tamago,0.9996	salmon,0.9978 tamago,0.9980 bentobox,0.5471	karaage,0.8931

With this example set of images, the results from the potato chip show that object detections using maximum level (Fig.32a) and medium level (Fig.32b) of brightness show completely correct result. The minimum level of brightness also

detects all of the ground truth positives but there is a false positive in which the model predicts an onigiri class (Japanese rice ball).

For the sushi set, the detection using maximum brightness level still gives 100% correct result. At a medium brightness level, there are three positives, with one of them being a false positive (bento box was detected), while the other two classifications are correct. The model fails to predict a positive result on ebi (shrimp) sushi, hence there is one false negative in the medium brightness level. Lastly, the results with minimum brightness level only produces one positive, which also turns out to be a false positive.

Based on these observations, the overall precisions and recalls are calculated by counting TP, FP, and FN of every images in the experiment. The results are tabulated and presented in Table 9 and Fig.34.

Table 9: Evaluation of the HTC model for 3 levels of brightness

Set	**Brightness level**	**Total objects**	**TP**	**FP**	**Total positives**	**FN**	**Precision (%)**	**Recall (%)**
Chip	Max	100	100	0	100	0	100.00	100.00
	Med		100	0	100	0	100.00	100.00
	Min		100	3	103	0	97.09	100.00
Sushi	Max	30	26	7	33	0	78.79	100.00
	Med		28	7	35	1	80.00	96.55
	Min		14	7	21	15	66.67	48.28

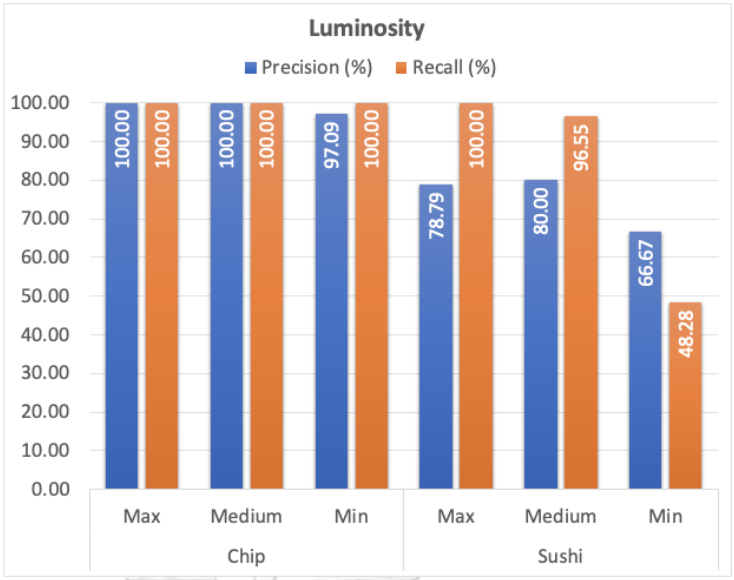


Fig. 34 Evaluation of the HTC model for 3 levels of brightness setting

As indicated by the result shown in Fig.34, luminosity only has a slight impact on the detection of the potato chip set. For the sushi set, maximum and medium level of brightness also give comparable results in precision and recall. However, when minimum level of brightness is used for a sushi set, there is a noticeable drop in both precision and, even more so, in the recall value. We can conclude that the lack of proper luminosity can affect the performance of this detection model, especially for objects similar to sushi, which come in different colors, shapes, and sizes.

4.1.3.2 Background Colors

Most of the images in the training set of our dataset contain white backgrounds. In this scenario, 6 different background colors are provided, including white, blue, green, orange, red, and yellow to see the effects that these background colors may have. Different color paper is attached to the wood plate base of the testing setup to perform this part of the experiment. For each background color, 10 photos are captured for the potato chip and sushi sets. The number of potato chips and sushi used per scene are the same as the previous scenario, hence there are a total of 100 potato chips and a total of 30 sushi pieces for the potato chips and sushi set, respectively. Examples of image and result are shown in Fig.35-38 and their prediction details are visualized in Table 10-11.

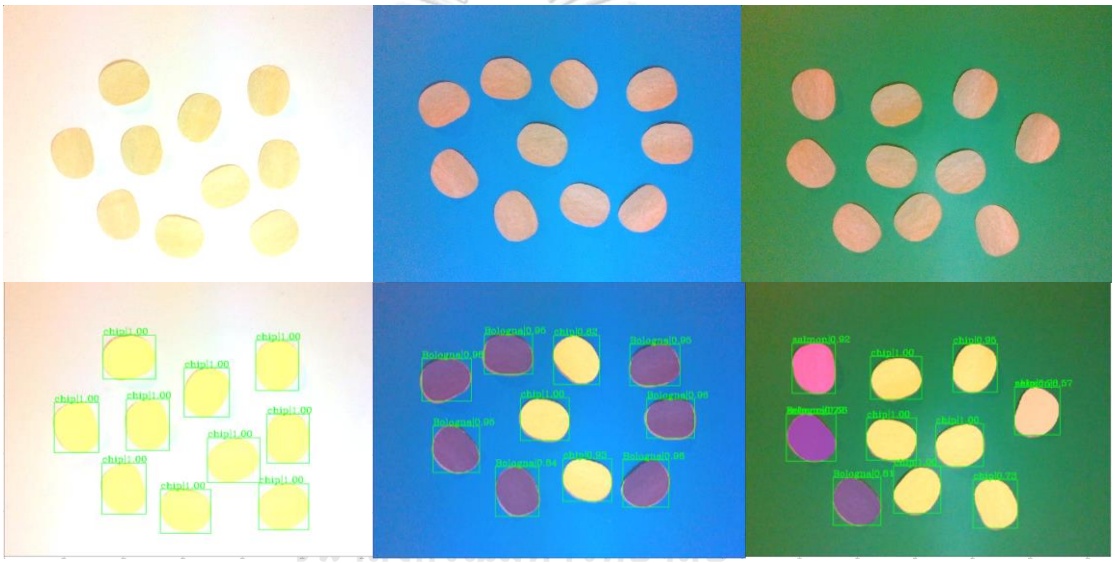


Fig. 35 Examples of potato chips detection results for different background colors: white (Left), blue (Middle), and green (Right)

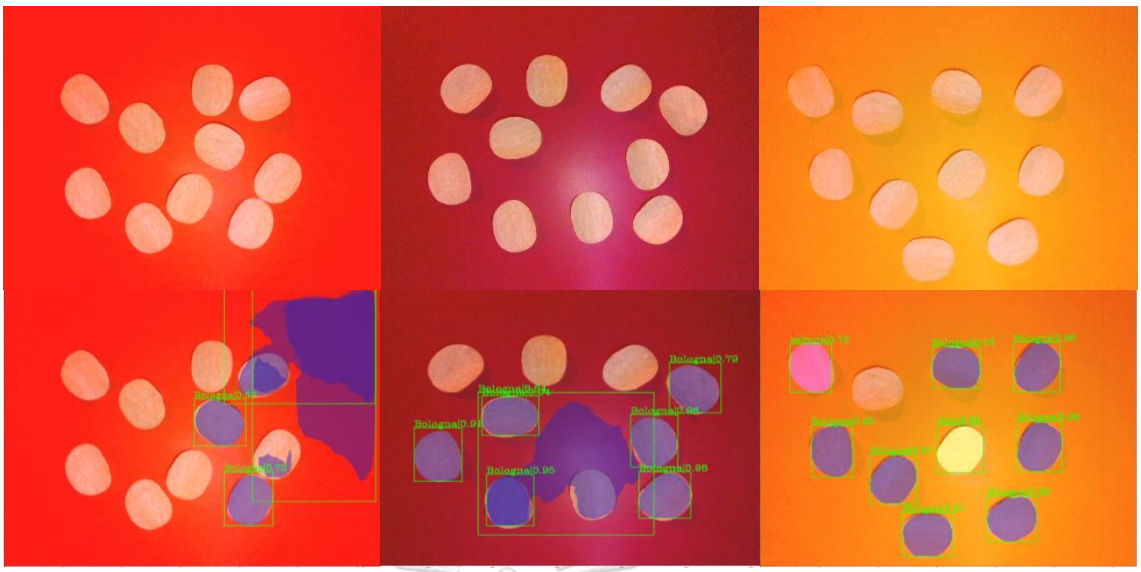


Fig. 36 Examples of potato chips detection results for different background colors: orange (Left), red (Middle), and yellow (Right)

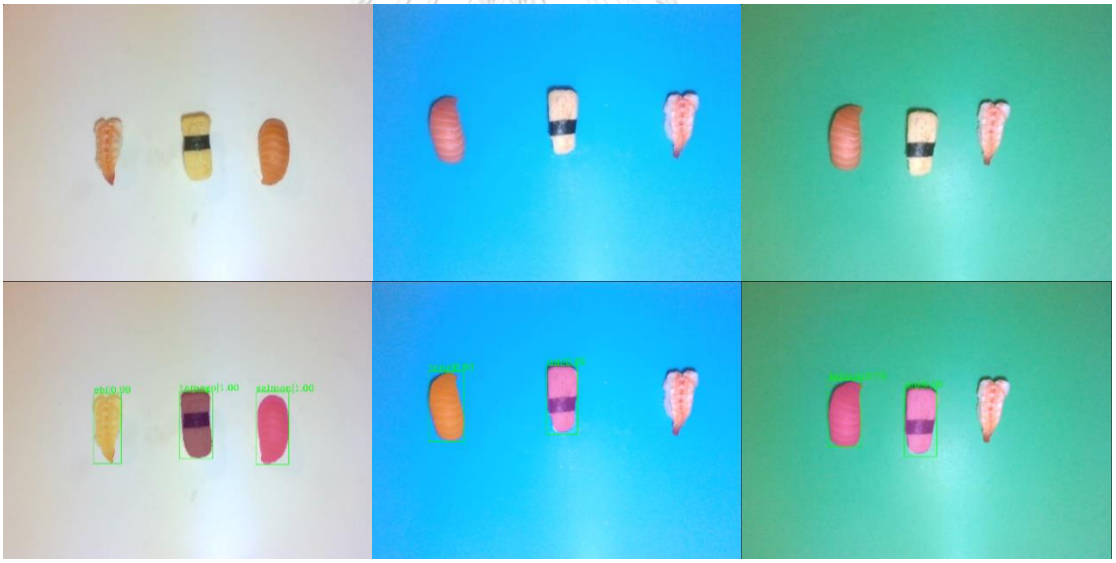


Fig. 37 Examples of sushi detection results for different background colors: white (Left), blue (Middle), and green (Right)

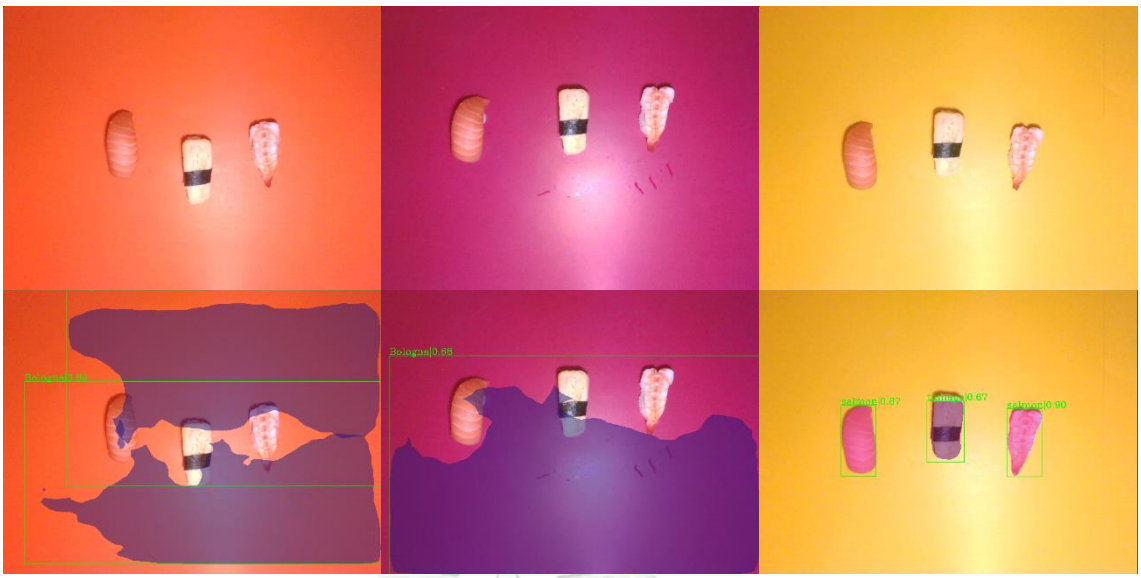


Fig. 38 Examples of sushi detection results for different background colors: orange (Left), red (Middle), and yellow (Right)

Table 10: Class predictions with IoU scores for 6 different background colors of potato chip set

White	Blue	Green	Orange	Red	Yellow
chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip, 0.9999 chip, 0.9989	chip,0.9983 chip,0.9344 chip,0.8168 bologna,0.9808 bologna,0.9798 bologna,0.9629 bologna,0.9536 bologna,0.9535 bologna,0.9491 bologna,0.8397	salmon,0.9160 salmon,0.7223 salmon,0.5746 chip, 0.9999 chip, 0.9998 chip, 0.9997 chip, 0.99818 chip, 0.9549 chip, 0.7584 chip, 0.7348 bologna,0.8103 bologna,0.5805	bologna, 0.7271 bologna, 0.5663 bologna,0.5414 bologna,0.5348	bologna,0.9782 bologna,0.9593 bologna,0.9474 bologna,0.9396 bologna,0.9075 bologna,0.8391 bologna,0.7936	salmon, 0.7343 chip, 0.7963 bologna, 0.9801 bologna, 0.9756 bologna, 0.9739 bologna, 0.9701 bologna, 0.8445 bologna, 0.8187 bologna, 0.7820

Table 11: Class prediction with IoU scores for 6 different background colors of sushi set

White	Blue	Green	Orange	Red	Yellow
ebi,0.9890 salmon,0.9998 tamago,0.9967	tuna,0.9311 uni,0.9233	salmon,0.7488 uni,0.9309	bologna,0.9269 bologna,0.8889	bologna,0.684	salmon,0.8977 salmon,0.8714 tamago,0.6672

As you can see from the above figures, the difference in background colors can cause a significant change in the detection performance of the models. To quantize these results, precision and recall values are tabulated in Table 12 and plotted in Fig. 39.

Table 12: Evaluation of the HTC model for different background colors

Set	Background colors	Total objects	TP	FP	Total positives	FN	Precision (%)	Recall (%)
Chip	White	100	100	0	100	0	100.00	100.00
	Blue		37	75	112	0	33.04	100.00
	Green		51	74	125	0	40.80	100.00
	Orange		0	41	41	80	0.00	0.00
	Red		0	64	64	46	0.00	0.00
	Yellow		11	77	88	13	12.50	45.83
Sushi	White	30	28	0	28	2	100.00	93.33
	Blue		3	17	20	14	15.00	17.65
	Green		15	14	29	5	51.72	75.00
	Orange		0	26	26	30	0.00	0.00
	Red		0	1	1	30	0.00	0.00
	Yellow		11	13	24	10	45.83	52.38

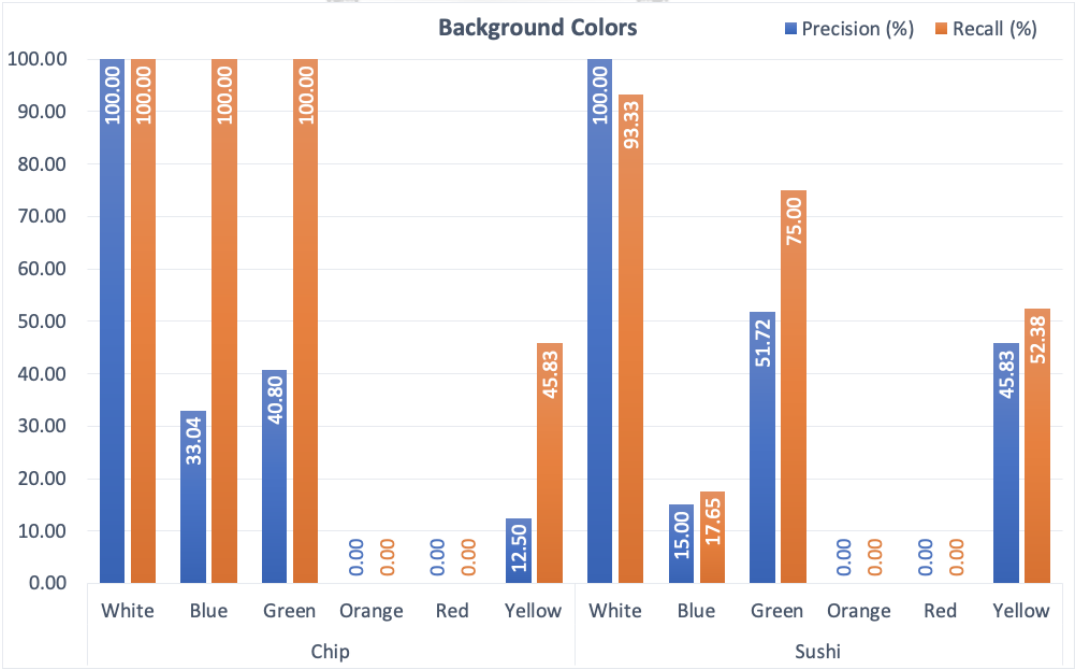


Fig. 39 Evaluation of the HTC model for different background colors

The results in Table 12 and Fig.39 show that the background colors severely affect the performance of the detection, causing the degradation of both precisions and recalls of the model. The results between the potato chip and sushi sets are also quite different because of the difference in object colors and textures under detection.

The precisions and recalls when using a white background color for detecting the 2 sets of food objects are the highest among all other colors. For both sets, the precisions and the recalls of orange and red backgrounds are zeros since there are no true positives predicted by the model. These two colors have the most serious effects on this detection model. One possible explanation might be that these two background colors are similar in color to the food object we are trying to detect such as ebi and salmon sushi. Moreover, these two colors might affect the auto white-balance adjustment of the camera. Another interesting observation is in the precision value of potato chip when yellow background is used. Compared to the result from the sushi set, the precision value for detecting a potato chip on a yellow background is lower than the case of sushi, which further reinforces the assumption that the similarity of color of the object being detected and the background color of the scene is may be to blame. However, the blue and green background affect the potato chip set in a different way. The recalls, or the ability to find positives, are at maximum values, but the precision shows a large drop. This means that the model detects all the potato chips as positives, but more than half of them is classified into wrong categories.

Based on the result, we can clearly see that background color in a scene can adversely affect the performance of our model in both the precision and recall values. One way to mitigate this problem is to add more images with different background colors into the training set, but this will certainly add to the overhead in the data-labeling step. Alternatively, for a control environment where the background of the scene remains more-or-less the same such as in a production line, we do not expect this to be a major problem as long as the background color of the training images are similar to the actual scene that the camera will capture. Being able to control some parts of the variables (i.e. background color) in the actual working environment can allow some requirements in the training data set to be relaxed, resulting in saved time and resources.

4.1.3.3 Addition of Non-Food Objects

In this scenario, we mix in non-food objects such as plastic plate, spoon, and chopsticks into a scene with the food objects to measure the performance of our detection model. This experiment is done on a white background color with maximum brightness level, and the added objects do not block or overlap with the food objects. The numbers of food objects used and the number of images taken are the same as the previous scenarios. For the potato chip set, we use total of 100 chips and for the sushi set, we use a total of 30 pieces of sushi.

The result and image examples from this testing scenario are shown in Fig.40 and the evaluation is shown in Table 13.

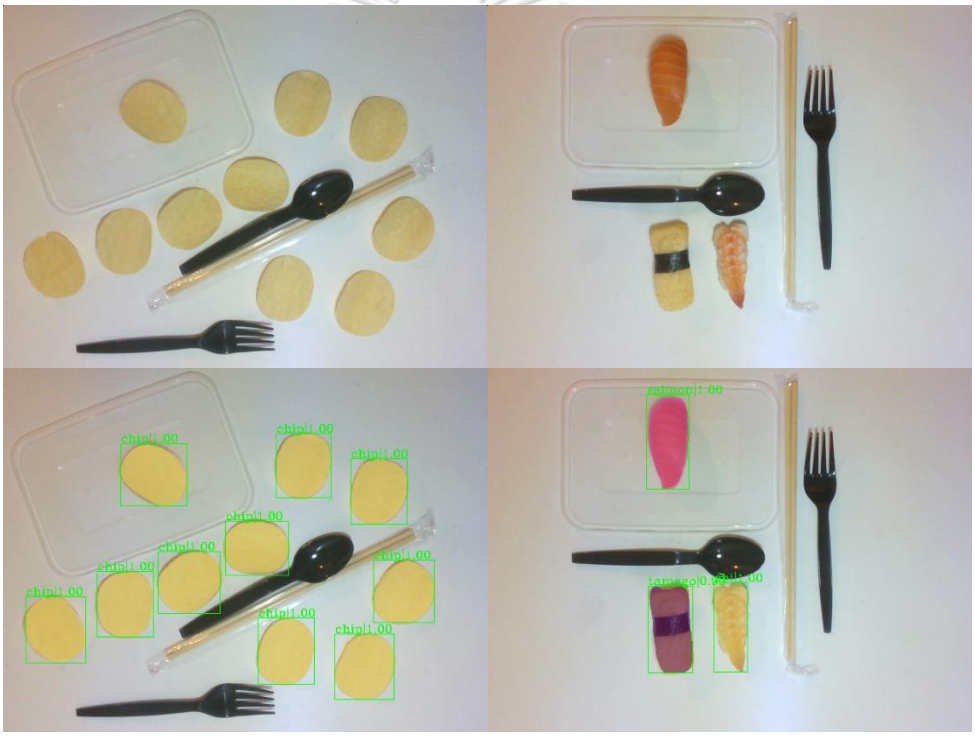


Fig. 40 Examples of results with addition of non-food objects scenario: potato chip set (Left) and sushi set (Right)

Table 13: Evaluation of the HTC model for addition of non-food objects scenario

Set	Total objects	TP	FP	Total positives	FN	Precision (%)	Recall (%)
Chip	100	100	0	100	0	100.00	100.00
Sushi	30	30	0	30	0	100.00	100.00

The results show that adding non-food objects into the scene does not seem to affect the precision and recall values, or the performance of the detection model like what has been observed in the case of luminosity and the background colors. This implies that this model can distinguish food objects from non-food objects, including the likes of plastic plate, plastic spoon, plastic folk, and chopsticks.

4.1.3.4 Placement

In this scenario, the food objects are arranged in three different configurations: normally-spaced, adjacent, and overlapping. For the overlapping configuration, we only consider the potato chip set is in our test case. The background is set to white and the luminosity is at the maximum level. We include 5 pieces of potato chips per image, and the number of sushi is kept the same as in the previous scenarios. The total number of images taken for each placement is 10. The image and detection result examples are shown in Fig.41-42 and their prediction details are shown Table 14-15.

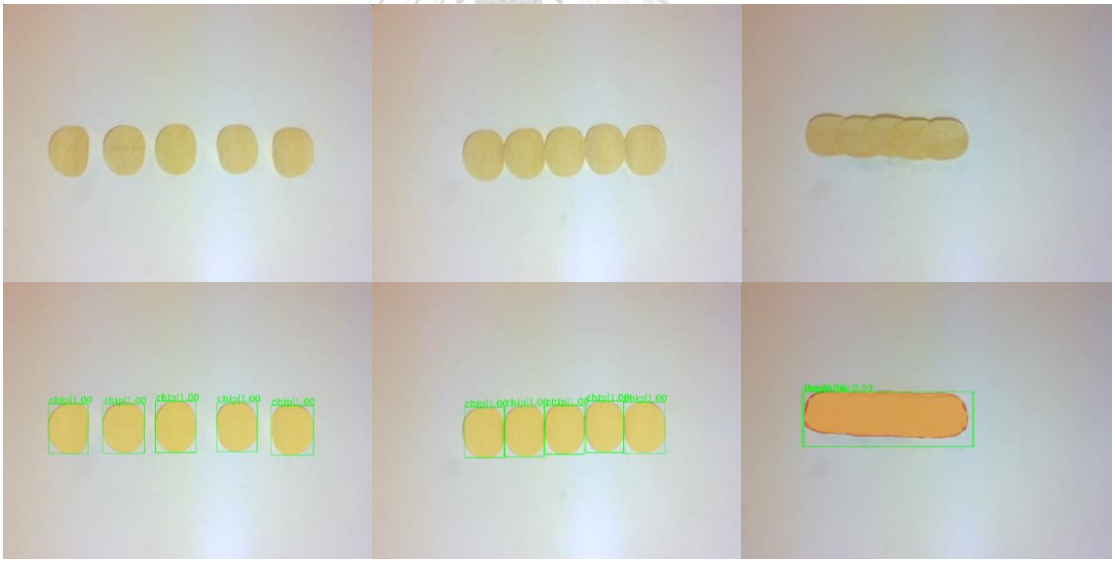


Fig. 41 Examples of potato chips detection results for different placements: normally spaced (Left), adjacent (Middle), and overlapping (Right)

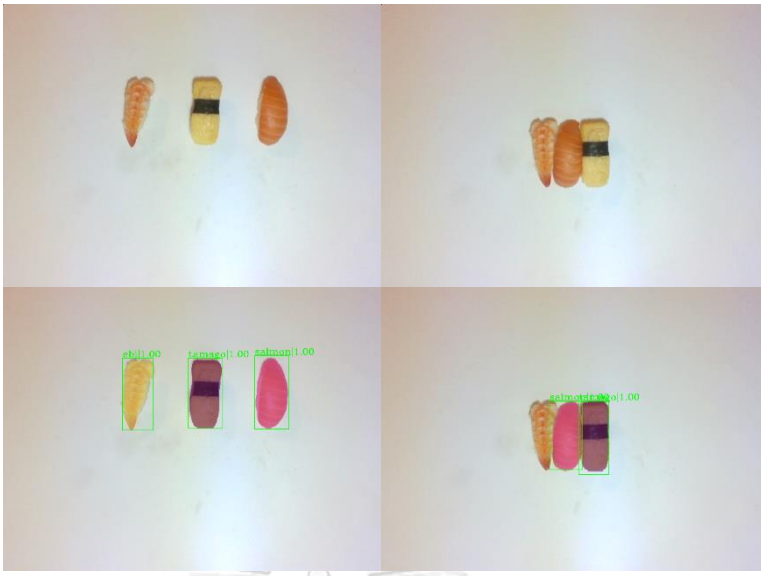


Fig. 42 Examples of sushi detection results for different placements: normally spaced (Left), and adjacent (Right)

Table 14: Class predictions with IoU scores for different placements of potato chip set

Normally spaced	Adjacent	Overlapping
chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999	chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9999 chip,0.9998	bentobox,0.5316 chip,0.7647

Table 15: Class predictions with IoU scores for different placements of sushi set

Normally spaced	Adjacent
ebi,0.9951 salmon,0.9999 tamago,0.9987	salmon,0.9994 tamago,0.9966

These examples show that the adjacent placement can have an effect on the prediction results. For example, in the adjacent placement of sushi objects, a false negative is present because the model fails to detect ebi sushi. In the case of an overlapping placement of potato chips, 2 false positives are observed: a bento box and a large chip which the model has mistakenly considered the 5 overlapping chips as one single chip. Consequently, there are also 5 false negatives in this case since none of the individual chips are correctly detected.

The detection result for this placement test is represented in Table 16 and Fig.43.

Table 16: Evaluation of the HTC model for different placements

Set	Placement	Total objects	TP	FP	Total positives	FN	Precision (%)	Recall (%)
Chip	Normally spaced	50	50	0	50	0	100.00	100.00
	Adjacent		50	0	50	0	100.00	100.00
	Overlapping		17	17	34	33	50.00	34.00
Sushi	Normally spaced	30	30	0	30	0	100.00	100.00
	Adjacent		25	3	28	2	89.29	92.59

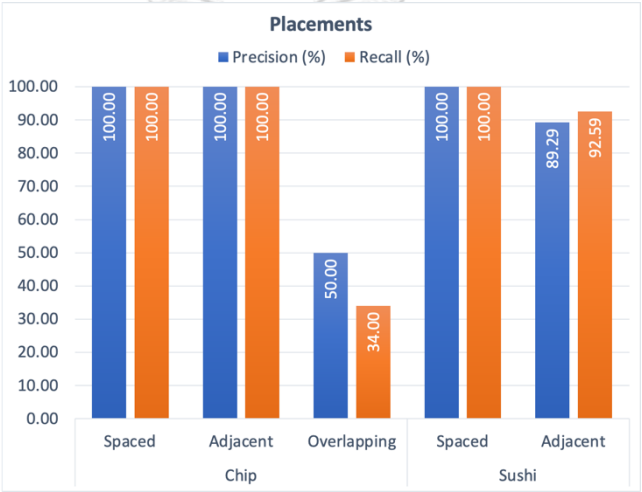


Fig. 43 Evaluation of the HTC model for different placements

This evaluation shows that the adjacent configuration affects the precision and recall of the sushi set by causing a ~10% decrease, but such degradation is not observed for the potato chip set. The overlapping configuration generates a 50% and a 66% decrease in the precision and the recall, respectively, for the potato chip set. The result suggests that the HTC model is able to successfully distinguish normally-spaced and adjacent objects, but it can be prone to error when it comes to detecting overlapping objects.

4.1.3.5 Incomplete objects

The next scenario that we test is to assess if the model can perform inference on an incomplete or partially-blocked objects. Only one piece of object is included per

image, and 30 photos were captured for each category of objects. Additionally, we also provide overall evaluation for the whole sushi set by combining the evaluation for each class. The background color is set to white and the brightness level is at maximum level. The image and detection result examples are shown in Fig.44-45. Then, we evaluate the model as shown in Table 17 and Fig.46.

Fig. 44 Examples of incomplete potato chip detection: normal image (Left), and detection (Right)

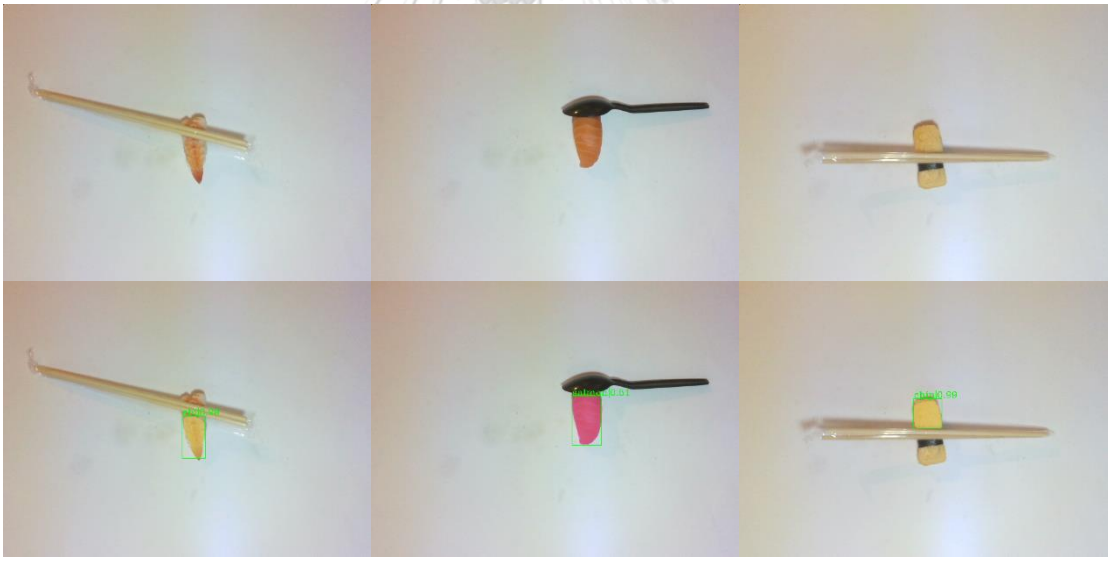


Fig. 45 Examples of incomplete sushi detection: ebi (Left), salmon (Middle), and tamago (Right)

Table 17: Evaluation of the HTC model for incomplete objects scenario

Set	Category	Total objects	TP	FP	Total positives	FN	Precision (%)	Recall (%)
Chip		30	30	2	32	0	93.75	100.00
Sushi	Ebi	30	13	5	18	12	72.22	52.00
	Adjacent	30	18	2	20	10	90.00	64.29
	Tamago	30	8	4	12	20	66.67	28.57
	Total	90	39	11	50	42	78.00	48.15

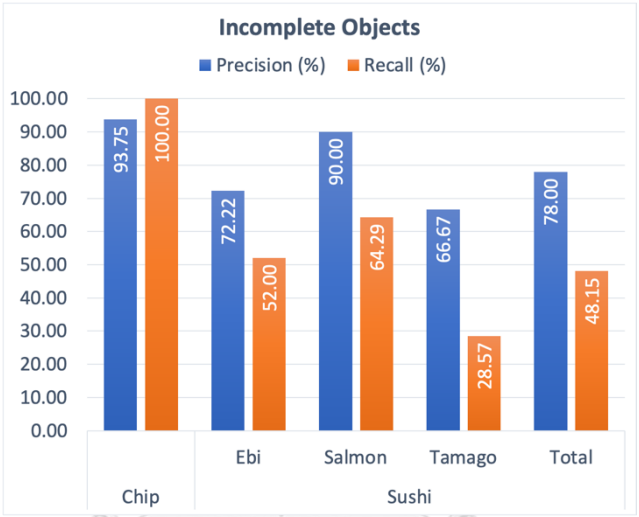


Fig. 46 Evaluation of the HTC model for incomplete objects scenario

The results show that the model can detect 100% of the incomplete potato chips with the accuracy of 93.75%. This high detection accuracy might be attributed to the fact that a potato chip has uniform color over the whole piece, even though its shape can be different.

In contrast, the result for the sushi data set under incomplete object testing scenario is relatively worse when compared to that of a potato chip. The least detected type of sushi is tamago (egg sushi) which is detected at only 28.57% recall value with 66.67% precision, and the most detected class of sushi is salmon, which has the recall and precision of 64.29% and 90.0%, respectively. And overall precision and recall of the sushi set is calculated to be 78.0% and 48.15%, respectively. Among these three types of sushi, salmon is inherently more uniform in color and texture as compared to the other two categories, which might explain why it has the highest precision and recall value in its class.

Depending on the situation, the ability to detect an incomplete object may or may not be beneficial to the system. For example, if the model is used as part of the robot perception system for picking and placing food object, it is undesirable to have the robot pick up incomplete or damaged object and place it in the final packaging. On the other hand, a defect-free object might appear to be incomplete due to being blocked by other objects. For this latter case, the failure to detect such object can result in an incorrect operation in the assembly line.

4.1.3.6 Image Resolution

In last scenario condition that we explore in the study is the effect of image resolution, we perform inference of 9 different possible streaming resolutions from the RGB-D camera, including, 320x180, 320x240, 424x240, 640x360, 640x480, 848x480, 960x540, 1280x720, and 1920x1080. Note that the background color is fixed to white color and the brightness is set to maximum level. We use 10 potato chips and 3 pieces of sushi. A total of 10 photos for each set and each image resolutions are captured and analyzed. Example images are shown in Fig.47-48 and the evaluation result is summarized in Table 18.

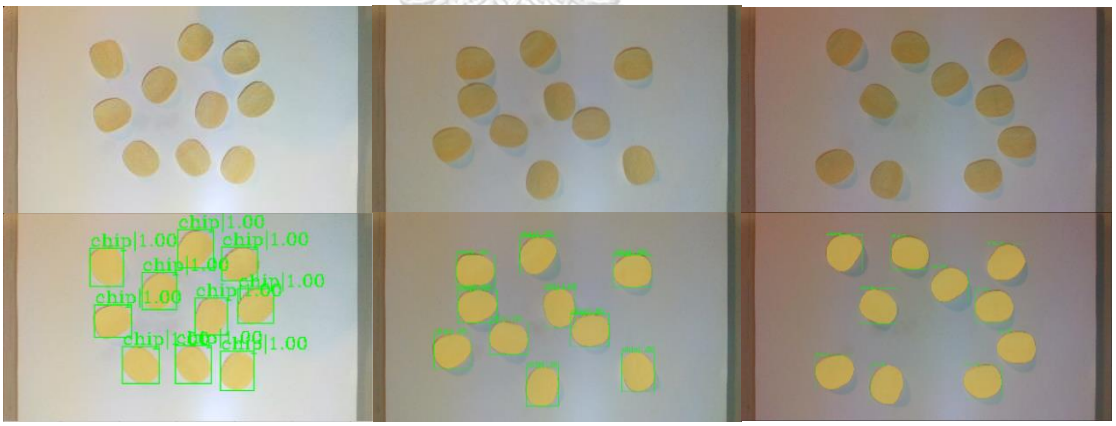


Fig. 47 Examples of the potato chip set detections in different image resolutions: 320x180 (Left), 848x480 (Middle), and 1920x1080 (Right)

Fig. 48 Examples of the sushi set detections in different image resolutions: 320x180 (Left), 848x480 (Middle), and 1920x1080 (Right)

Table 18: Evaluation of the HTC model for different image resolutions

Set	Placement	Total objects	TP	FP	Total positives	FN	Precision (%)	Recall (%)
Chip	320x180	100	100	0	100	0	100.00	100.00
	320x240		100	0	100	0	100.00	100.00
	424x240		100	0	100	0	100.00	100.00
	640x360		100	0	100	0	100.00	100.00
	640x480		100	0	100	0	100.00	100.00
	848x480		100	0	100	0	100.00	100.00
	960x540		100	0	100	0	100.00	100.00
	1280x720		100	0	100	0	100.00	100.00
	1920x1080		100	0	100	0	100.00	100.00
Sushi	320x180	30	30	0	30	0	100.00	100.00
	320x240		30	0	30	0	100.00	100.00
	424x240		30	0	30	0	100.00	100.00
	640x360		30	0	30	0	100.00	100.00
	640x480		30	0	30	0	100.00	100.00
	848x480		30	0	30	0	100.00	100.00
	960x540		30	0	30	0	100.00	100.00
	1280x720		30	0	30	0	100.00	100.00
	1920x1080		29	0	29	1	100.00	96.67

According to the evaluation in Table 18, we can conclude that different image resolutions show little to no effect on the precision and recall values of the HTC model.

Since the images used in the training of the model are of low to medium resolutions, this implies that detection of high-resolution test images is still possible without the need to include high resolution images in the training data set.

Finally, we have been through all six test scenarios and observe how each one of them can impact the performance of the HTC model in our experimental setup. We have visualized the results and discussed them above. This would be beneficial to researchers who want to improve an object detection or instance segmentation to use in food automation. Next, we would like to show the performance of our trained models on different conditions.

4.1.4 Inference Speed Analysis

In this section, we compare inference speed between the 2 models: Cascade Mask R-CNN and HTC without semantic segmentation. Two testing conditions are used in the comparison, namely, the number of objects presented in the image and resolution of the image being analyzed. The models used are the 300-epoch models that are trained on Google Colab. The testing is also completely done on Colab with Nvidia Tesla T4 GPU. We measure the inference speed by calculating the difference in time between before and after we execute the function, inference_detector, which is the function provided by the MMDetection toolbox, and is used to perform inference in the previous section.

4.1.4.1 Number of objects

We perform the evaluation on the potato chip class by preparing 20 different images of potato chips where the number of potato chips in an image varies from 1 to 20 potato chips. The inference is performed 30 times for each image and the average time is calculated to be the representative value for that particular image. The representative values of the inference speed (in frame per seconds) for Cascade Mask R-CNN and HTC model are shown in Fig.49.

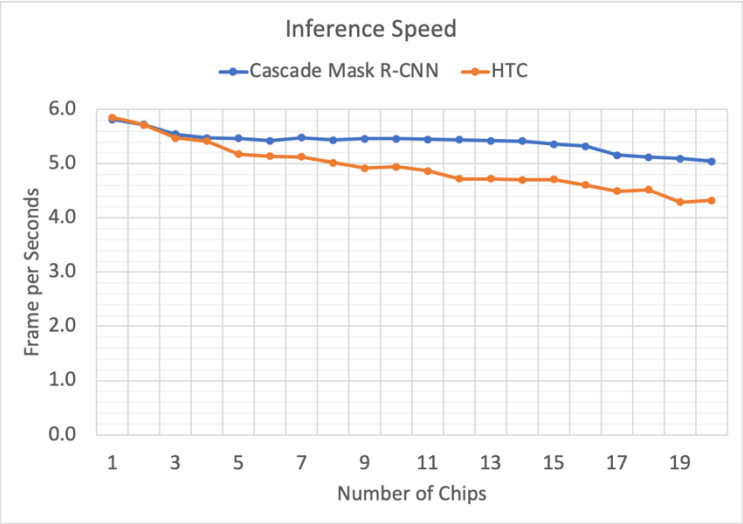


Fig. 49 Inference speed comparison between Cascade Mask R-CNN model and HTC model on different numbers of potato chips in an image

As is evident in the figure, the inference speeds of both models tend to decrease as the number of objects increases, as expected. The HTC model decreases at a slightly faster rate than that of Cascade Mask R-CNN model. However, these two models share approximately the same values when the number of objects is small (4 objects or less in this experiment).

4.1.4.2 Image Resolution

In this section, we perform inference for different resolutions of potato chip images, where each image contains 10 potato chips. For each image resolution, there are 10 different images. We conduct inference on all of the images, 30 times per image. We find the average values of 30 iterations and then calculate the final average values of among the 10 images. There are 9 different possible streaming resolutions including, 320x180, 320x240, 424x240, 640x360, 640x480, 848x480, 960x540, 1280x720, and 1920x1080. Most of them are in 16:9 scale, except for 320x240 and 640x480, which are in 4:3 ratio.

The inference speed on different image resolutions with Cascade Mask R-CNN and HTC modes is shown in Fig.50.

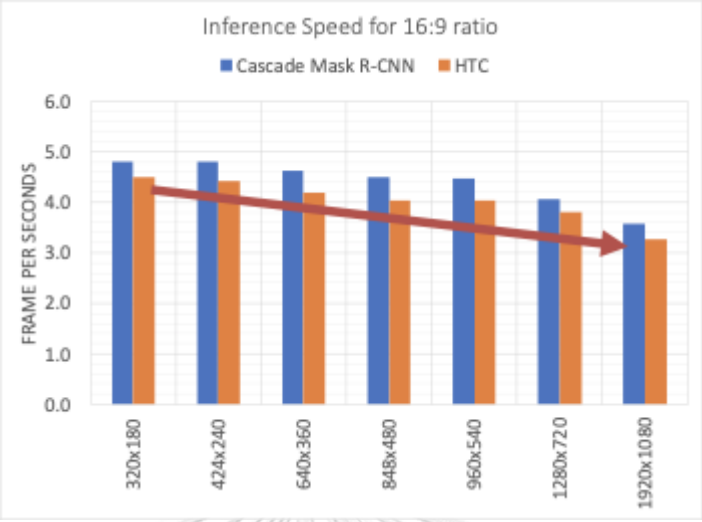


Fig. 50 Inference speed comparison between Cascade Mask R-CNN model and HTC model on 16:9 image resolutions

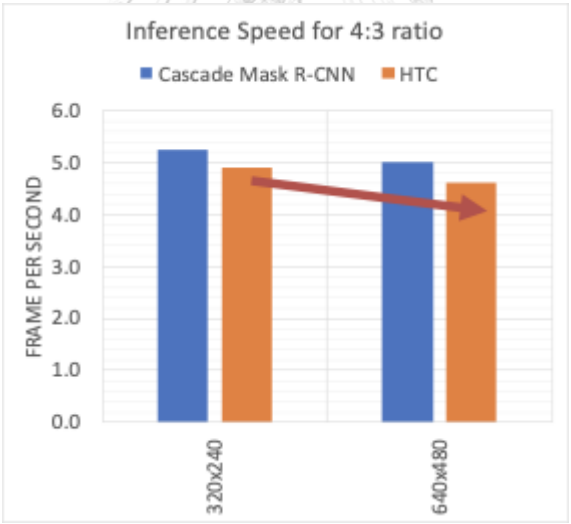


Fig. 51 Inference speed comparison between Cascade Mask R-CNN model and HTC model on 4:3 image resolutions

From the charts, the inference speed values tend to go down as the resolution, or the number of pixels, increases. Moreover, the inference speeds of Cascade Mask R-CNN are higher than HTC for all image resolution tested here. Additionally, the result is consistent for both 16:9 and 4:3 ratio images.

We have measured and compared the inference speeds using two test conditions. We can conclude that the Cascade Mask R-CNN model generally shows a faster inference speed than the HTC model in a given condition. The results also indicate that the number of objects in an image affects the speed of the detection in both models. Additionally, the higher number of pixels or higher resolution will require more time to perform inference, and a 4:3 ratio images seem to have faster inference rate than 16:9 ratio images even if they possess higher number of pixels.

In this section, we have discussed all of the 2D segmentation results, namely, model evaluation, segmentation results on test set, different scenario impacts on HTC model performance, and inference speed analysis. The result illustrates the ability of the model to detect food objects in many different conditions, but it also highlights the limitations and issues that occur in other cases. Nevertheless, the result and discussion presented here can serve as a guide towards improving 2D instance segmentation models in the future.

4.2 3D Point cloud extraction

In this part, we will discuss our 3D point cloud extraction results and compare the two extraction methods that are used.

4.2.1 Point Cloud Extraction Results

Fig. 51 (Left) shows an RGB frame of the depth camera and Fig. 51 (Right) shows the corresponding extracted point cloud of that frame with the depth cloud of the surroundings. Using rviz for visualization, the point cloud messages of each object are published in different ROS topics with different colors and they can be visualized simultaneously. In this example image, there are 4 objects being detected: Japanese lunch box (blue), Japanese rice ball (green), potato chip (yellow), and Japanese fried chicken (red).

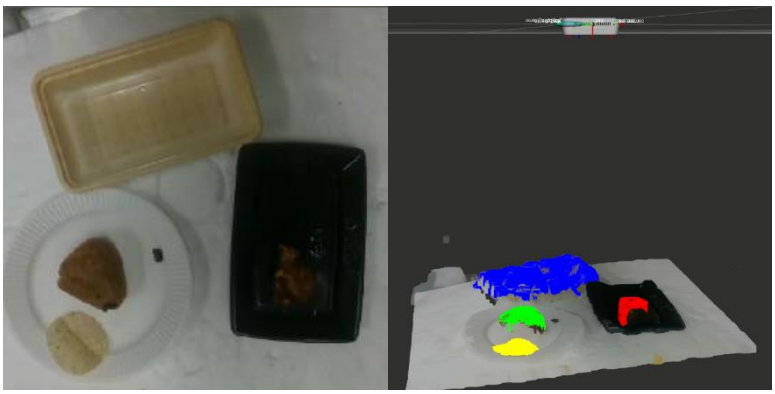


Fig. 52 RGB frame of the depth camera (Left) and extracted point cloud visualization (Right)

Fig.52 shows the extracted point cloud of a Japanese rice ball (onigiri) as seen from different perspectives. The left column shows the RGB images and the right column shows their corresponding point cloud information. The green cluster represents the point cloud of the detected object (onigiri) and the white cluster represents the depth information of the surroundings (plate and table). The point cloud information of each object is streamed in real-time as the camera raw data.

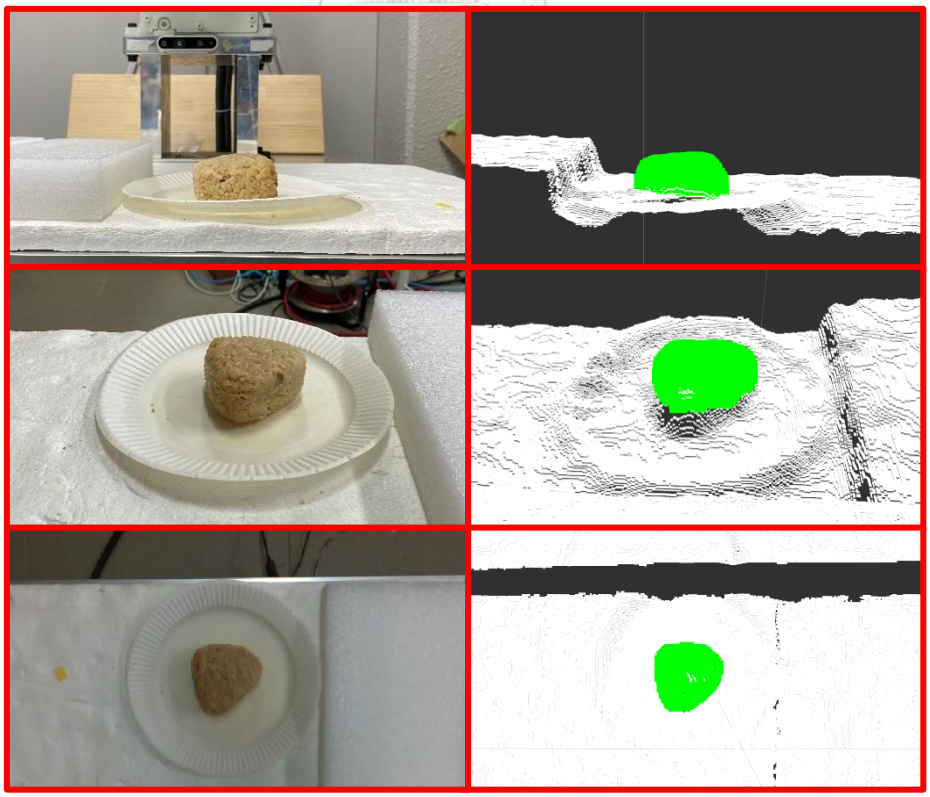


Fig. 53 Japanese rice ball point cloud visualization

4.2.2 Extraction Method Comparison

Comparing between the two methods of point cloud extraction, the first extraction method using depth topic shows a faster processing time as compared to the second extraction method, which uses point cloud topic. The processing time comparison of their first thousand frames is shown in Fig 53. Measured using 'timeit' Python library, the average processing time of the first and the second extraction methods are 0.149 and 0.211 seconds (6.71 and 4.74 fps), respectively. This translates to about a 1.4x faster processing time for the first extraction method. It is worth noting that although the second extraction method has worse performance, it is easier to understand and implement. Furthermore, the point cloud extracted using the second method comes with RGB information of the corresponding RGB frame, which could be beneficial to other types of analysis as well.

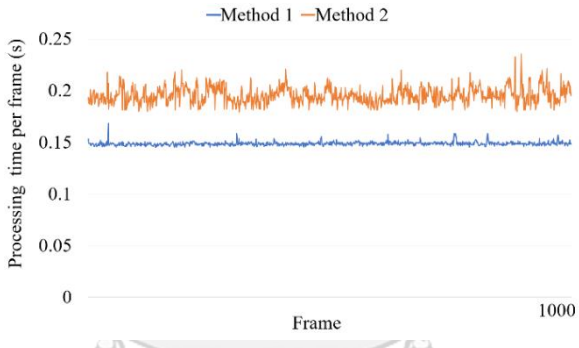


Fig. 54 Comparison of processing time between two methods of point cloud extraction

CHAPTER 5 CONCLUSION

Food industry will continue to evolve as technology becomes more integrated as part of the daily operations to improve efficiency in productions, streamline distribution, and ensure quality control. Manual and repetitive tasks carried out by humans will gradually be replaced by automated machines and industrial robots. To this end, we anticipate that machine vision will play a key role in enabling such a transition. In this work, we have implemented instance segmentation models using Cascade Mask R-CNN and HTC models that are able to detect Japanese food using 2D RGB data. The 2D segmentation result is then combined with a point cloud information acquired using depth sensing camera to create a 3D segmentation that is essential for accomplishing tasks that involve grasping and picking-and-placing object.

The segmentation result between Cascade Mask R-CNN and HTC models is evaluated. HTC model consistently shows higher mean average precision score and less false positive when compared to Cascade Mask R-CNN, but its inference speed tends to be slower, especially as more objects are added to the scene. In addition, we evaluate the performance of the HTC model under different scenario conditions, including luminosity, background colors, placement, resolution, addition of non-food objects, and having an incomplete object. In our experiment, it is observed that luminosity, background colors, and having an incomplete object show the most impact on the precision and recall values of an image segmentation.

Two methods for point cloud extraction using depth sensing camera are presented. One is through subscription of a depth topic and the other is through point cloud topic. The depth topic method is able to achieve 6.71 fps, which is 1.4x higher frame rate as compared to the point cloud topic, but it also requires more post-processing calculation. Once depth information is combined with previously segmented 2D result, a centroid of each 3D segmented object can be calculated by taking the average value of the x, y, and z coordinates.

Our study shows the feasibility of using 2D RGB data for classifying and localization of complex, non-uniform food objects. To implement such system in the actual production line, careful consideration in regards to the environment must be

taken. As discussed previously, the backgrounds and surroundings of where the food objects are situated can have a significant impact on the classification. If the background and the environment can be well-controlled, it would reduce the burden of having to train the classification model on many different backgrounds, for example. A detection model tailored for a specific usage environment, instead of a universal one, would require less dataset to train without compromising on its real-world usage accuracy. In addition, instance segmentation does not necessarily have to rely only on 2D RGB data. In fact, it should be possible to improve the performance of the segmentation model by incorporating depth information into the training process, or use it to isolate target objects from the background itself.

REFERENCES

[1] P. Hoden, "Automation in the food industry," in *Automation in the food industry* vol. 2020, ed, 2011.

[2] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into High Quality Object Detection," 12/03 2017.

[3] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 22-29 Oct. 2017 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, 06/04 2015, doi: 10.1109/TPAMI.2016.2577031.

[5] K. Chen *et al.*, "MMDetection: Open MMLab Detection Toolbox and Benchmark," *arXiv e-prints,* p. arXiv:1906.07155, 2019.

[6] A. Zhang, Z. C. Lipton, and M. L. a. A. J. Smola, *Dive into Deep Learning*. 2020.

[7] S. Halbe, "Object Detection and Instance Segmentation: A detailed overview," ed, 2020.

[8] K. Chen *et al.*, "Hybrid Task Cascade for Instance Segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 15-20 June 2019 2019, pp. 4969-4978, doi: 10.1109/CVPR.2019.00511.

[9] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*. 2016, pp. 770-778.

[10] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.,* pp. 1-42, 01/01 2014.

[11] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv 1409.1556,* 09/04 2014.

[12] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *University of Toronto,* 05/08 2012.

[13] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, pp. 1-1, 05/24 2016, doi: 10.1109/TPAMI.2016.2572683.

[14] S.-H. Tsang, "Review: FCN — Fully Convolutional Network (Semantic Segmentation)," ed, 2018.

[15] "Beginner's guide to depth." https://www.intelrealsense.com/beginners-guide-to-depth (accessed Sep 20, 2020).

[16] "What is Apple Face ID and how does it work?" https://www.pocket-lint.com/phones/news/apple/142207-what-is-apple-face-id-and-how-does-it-work (accessed Nov 30, 2020).

[17] "COMPARING DRONE LIDAR AND PHOTOGRAMMETRY." https://terra-drone.eu/en/articles-en/comparing-drone-lidar-and-photogrammetry (accessed Nov 30, 2020).

[18] F. Garcia *et al.*, "ANALYSIS OF LIDAR SENSORS FOR NEW ADAS APPLICATIONS. USABILITY IN MOVING OBSTACLES DETECTION," 01/01 2009.

[19] "TensorBoard: TensorFlow's visualization toolkit." https://www.tensorflow.org/tensorboard (accessed Nov 30, 2020).

[20] "Why TensorFlow." https://www.tensorflow.org/about/case-studies (accessed Nov 30, 2020).

[21] D. Mwiti, "Deep Learning with PyTorch: An Introduction," ed, 2018.

[22] A. Paszke *et al.*, "Automatic differentiation in PyTorch," 2017.

[23] Y. Tawil, "An Introduction to Robot Operating System (ROS)," ed, 2017.

[24] "ROS Concepts." http://wiki.ros.org/ROS/Concepts (accessed Oct 21, 2020).

[25] "SIA5F 7-Axis Articulated Arm." https://www.motoman.com/enus/products-/robots/industrial/assembly-handling/siaseries/sia5f/ (accessed Apr 20, 2020).

[26] "Intel RealSense Depth Camera D435i datasheet." https://www.intelrealsense.com-/wpcontent/uploads/2020/05/Intel-RealSense-D400-SeriesDatasheet-May-2020.pdf (accessed Mar 15, 2020).

[27] "Nvidia GeForce RTX 2080 Ti Graphics Card." https://www.nvidia.com/en-us/geforce/graphicscards/rtx-2080-ti/ (accessed Mar 15, 2020).

[28] E. Bisong, "Google Colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, E. Bisong Ed. Berkeley, CA: Apress, 2019, pp. 59-64.

[29] "Instance Segmentation on COCO test-dev." https://paperswithcode.com/sota/instance-segmentation-on-coco (accessed Feb 9, 2020).

[30] A. Eklund, "Cascade Mask R-CNN and Keypoint Detection used in Floorplan Parsing," Independent thesis Advanced level (professional degree) Student thesis, UPTEC IT, 20029, 2020. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-415371

[31] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context," 05/01 2014.

[32] J. Brooks. "COCO Annotator." https://github.com/jsbroks/coco-annotator/ (accessed 2020).

[33] J. Hui. "mAP (mean Average Precision) for Object Detection." https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173#:~:text=mAP%20(mean%20average%20precision)%20is,difference%20between%20AP%20and%20mAP. (accessed Nov 25, 2020).

VITA

NAME	Suthiwat Yarnchalothorn
DATE OF BIRTH	16 July 1996
PLACE OF BIRTH	Lopburi
INSTITUTIONS ATTENDED	Chulalongkorn University
HOME ADDRESS	688/134 Soi Phayanak, Phayathai Road, Thanon Petchburi, Ratchathewi, Bangkok 10400.