OBJECT DETECTION IN INTELLIGENT BILLING SYSTEM FOR CONVEYOR BELT SUSHI RESTAURANT

Mr. Rangrak Maitriboriruks

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Department of Computer Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2021

การตรวจหาวัตถุในระบบวางบิลอัจฉริยะสำหรับร้านซูชิสายพาน

นายรังรักษ์ ไมตรีบริรักษ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Thesis Title	OBJECT DETECTION IN INTELLIGENT BILLING SYSTEM FOR CONVEYOR BELT SUSHI RESTAURANT
By	Mr. Rangrak Maitriboriruks
Field of Study	Computer Science
Thesis Advisor	Associate Professor Yachai Limpiyakorn, Ph.D.

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

.... Dean of the FACULTY OF ENGINEERING

(Professor SUPOT TEACHAVORASINSKUN, D.Eng.)

THESIS COMMITTEE

.... Chairman

(Assistant Professor SUKREE SINTHUPINYO, Ph.D.)

.... Thesis Advisor

(Associate Professor Yachai Limpiyakorn, Ph.D.)

.... External Examiner

(Paskorn Apirukvorapinit, Ph.D.)

รังรักษ์ ไมตรีบริรักษ์ : การตรวจหาวัตถุในระบบวางบิลอัจฉริยะสำหรับร้านซูชิสายพาน. (OBJECT DETECTION IN INTELLIGENT BILLING SYSTEM FOR CONVEYOR BELT SUSHI RESTAURANT) อ.ที่ปรึกษาหลัก : รศ. ดร.ญาใจ ลิ่มปิยะกรณ์

องค์กรต้องดำเนินการอัตโนมัติทุกที่และทุกเวลา โดยเฉพาะอย่างยิ่งในช่วงที่ชีวิตประจำวันเปลี่ยนแปลงไปทั่วโลก แนวโน้มการใช้เทคโนโลยี โดยเฉพาะ ปัญญาประดิษฐ์ ได้กลายเป็นปัจจัยสำคัญที่ทำให้เกิดนวัตกรรมที่ก่อกวน วิทยานิพนธ์นี้จึงนำเสนออินเทอร์เฟซการเขียนโปรแกรมแอปพลิเคชันของเครื่องตรวจจับวัตถุที่ใช้งานกับ โยโลวี4 และโอเพนซีวีเพื่อจำแนกราคาของจานซูชิแยกตามสี เครื่องตรวจจับวัตถุเป็นส่วนหนึ่งของแอปพลิเคชันมือถือข้ามแพลตฟอร์มอัจฉริยะเพื่ออำนวยความสะดวกในกระบวนการเรียกเก็บเงินสำหรับธุรกิจซูชิสายพาน ส่วนหน้าของระบบได้รับการพัฒนาด้วยฟลัตเตอร์เพื่อสร้างโค้ดเบสเดียวสำหรับส่วนต่อประสานกับผู้ใช้ เพื่อจัดการกับสีต่างๆ ของภาพที่เกิดจากการใช้กล้องมือถือที่แตกต่างกัน การถ่ายโอนสีจะใช้สำหรับการถ่ายโอนสีของชุดข้อมูลภาพไปยังภาพที่ผู้ใช้ถ่าย สถาปัตยกรรมไมโครเซอร์วิสถูกนำมาใช้สำหรับส่วนหลังของระบบการประสานกันของ โยโลวี4, โอเพนซีวี และ สปริงบูตเรสต์ เอพีไอ จะสร้าง เอพีไอ เพื่อคำนวณค่าอาหาร สร้างรหัส คิวอาร์ สำหรับชำระบิล และรักษาผลประโยชน์การเป็นสมาชิกของลูกค้า แบบจำลองการตรวจจับวัตถุที่สร้างขึ้นค่าความเที่ยงตรง 97%, การเรียกกลับ 97%, คะแนนเอฟหนึ่ง 97% และ เอ็มเอพี 97.3% ระบบการเรียกเก็บเงินอัจฉริยะที่นำเสนอในงานนี้จะช่วยเร่งเวิร์กโฟลว์ เพิ่มผลผลิต ลดของเสีย และขับเคลื่อนการเคลื่อนไหวเพื่อสังคมไร้สัมผัส

สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์	ลายมือชื่อนิสิต ....
ปีการศึกษา	2564	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....

6370245421 : MAJOR COMPUTER SCIENCE
KEYWORD: object detection, conveyor belt sushi, intelligent process automation, smart billing

Rangrak Maitriboriruks : OBJECT DETECTION IN INTELLIGENT BILLING SYSTEM FOR CONVEYOR BELT SUSHI RESTAURANT. Advisor: Assoc. Prof. Yachai Limpiyakorn, Ph.D.

Organization must automate wherever and whenever they can, particularly during today’s global changes in daily lifestyles. Trends regarding the use of technology, especially AI has emerged as a key enabler for disruptive innovation. This thesis thus presents the application programming interface of object detector implemented with YOLOv4 and OpenCV for classifying the prices of sushi plates distinguished by colors. The object detector is part of the smart cross-platform mobile application to facilitate billing process for conveyor belt sushi business. The frontend is developed with Flutter to build single codebase for UIs. To handle the variants of image colors resulting from the use of different mobile cameras, color transfer is used for transferring the image dataset colors to images captured by users. Microservices architecture is adopted for the backend. Orchestration of YOLOv4, OpenCV and Spring Boot REST API will create APIs to calculate food cost, generate QR code for bill payment, and maintain customer membership benefits. The constructed object detection model achieved the precision of 97%, recall of 97%, F1-score of 97% and mAP of 97.3% The smart billing system presented in this work would accelerate the workflow, increase productivity, reduce waste and drive moving for contactless society.

Field of Study:	Computer Science	Student's Signature ....
Academic Year:	2021	Advisor's Signature ....

ACKNOWLEDGEMENTS

The completion of this thesis could not have been possible without the assistance and participation of so many people whose names may not all be enumerated. Their contributions are sincerely appreciated and gratefully acknowledged. However, the group would like to express deep appreciation and indebtedness particularly to the following.

Assoc. Prof. Dr. Yachai Limpiyakorn, Asst. Prof. Dr. Sukree Sinthupinyo and Dr. Paskorn Apirukvorapinit for their time, endless effort and generous advice during the presentation.

And thanks a lot to Assoc. Prof. Dr. Yachai Limpiyakorn for being advisor, who is always helping to give advice for thesis and following the work process, is a person who gives good advice in every matter. causing students to choose the right path. And thank you to my family for always supporting me when I need encouragement. Last but not least, to all relatives, friends, family others who in one way or another shared their support, either morally, financially and physically, thank you.

Rangrak Maitriboriruks

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
Introduction

1.1 Statement of the problems

It has been widely known that conveyor belt sushi experienced is a big boom in popularity in Japan. The main reasons are a wide variety of menu choices (example as shown in Figure 1), reasonable price, and accessible to all types of customers. Especially when people loved being able to eat quickly. Therefore, Japanese people decide to choose sushi served on rotation plates. The main concept of a typical conveyor belt sushi restaurant is using color-coded plates to identify sushi price and calculate the bill. The color-coded plate does not limit to only sushi item, non-sushi items can be calculated as well, as shown in Figure 2. However, the main problem with conveyor belt sushi restaurant is the method for calculating the bill. Generally, customers call a staff member to count all the sushi plates on the table, where each sushi plate has its own color-coded. If there are many customers who come to the restaurant and order a variety of sushi items with various prices, this may cause a delay in sorting and counting the plates. Repetition and fatigue may cause human error in the restaurant. From the aforementioned factors, it impacts delays in services. When service failure occurs due to a long waiting time, this reflects negatively on customer satisfaction and the store's image.

This research presents the application of Object Detection to count the number of color-coded plates. The study is based on the patterns of sushi color-coded plates from the most famous conveyor belt sushi restaurant in Japan. Central World has the first branch of conveyor belt sushi from Japan that opened its door in Thailand [1]. Applying the color-coded detection of sushi plates by implementing YOLO technique version 4 (YOLOv4) in conjunction with the OpenCV Library. The input data first use the photos from the mobile camera of the restaurant staffs. Then processing the color-coded plates recognition, and calculating the amount that the customer has to pay. It is designed and developed as an Application Programming Interface (API) developed with Java framework, Spring Boot REST API. This tool can work on 3 platforms, namely Android, iOS and Web application. It is expected that the results of the research will enhance quality of service, operational efficiency, responsive calculation, paper usage and human accuracy. Additionally, reducing queue at the counter in front of the restaurant and keeping customers safe from contracting Covid 19 in the epidemic situation of emerging infectious disease.

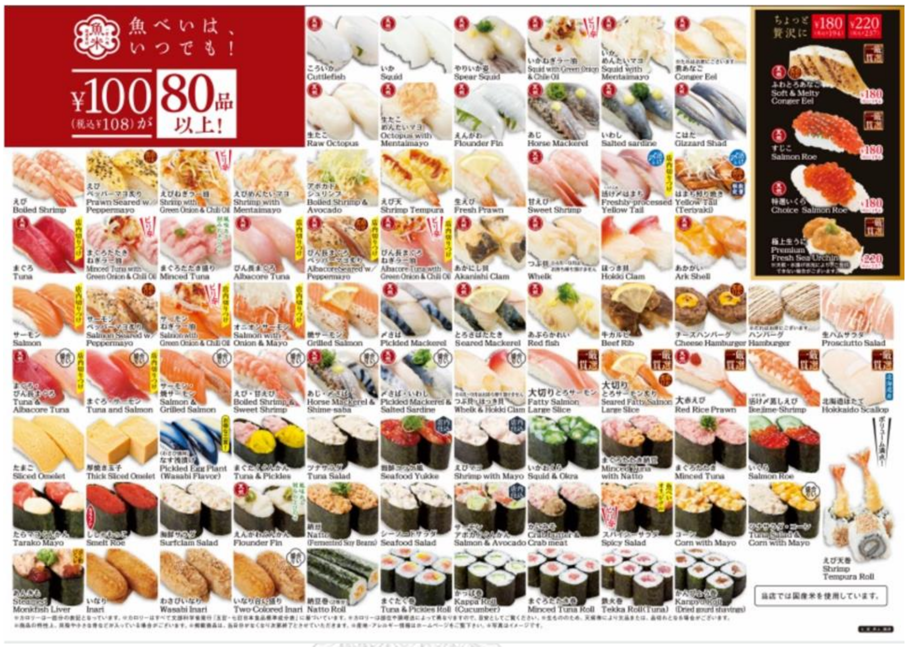


Figure 1: Uobei Sushi Menu [2]

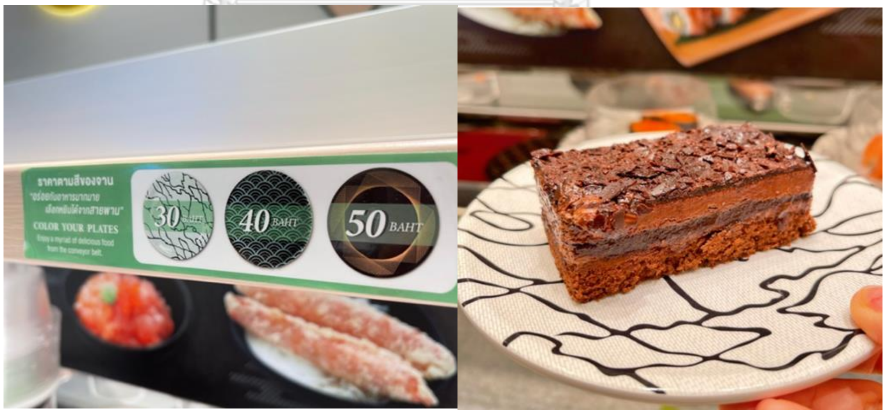


Figure 2: Price of desserts on sushi plates [3]

1.2 Objectives

Introducing an Intelligent billing system for conveyor belt sushi restaurants. by applying technology to detect objects from mobile device shot of sushi plates. To reduce the time for counting containers and calculating the cost of food that customers have to pay. as well as supporting a new normal life in a communicable disease epidemic situation.

1.3 Scope of Study

1.3.1 Using the dataset of sushi plates from the Japanese's most famous conveyor belt sushi franchise restaurant in Bangkok, Thailand.

1.3.2 Able to detect objects, consisting of free water cups, sauce dishes, sushi dishes in 4 colors: 1. Red, 40 baht, 2. Silver, 60 baht, 3. Gold, 80 baht, and 4. Black, 120 baht , and the container is not sushi dishes.

1.3.3 The picture of the sushi plate is clear. Not far away and the image is not broken

1.4 Research Methodology

1.4.1 Study and research related theories and literature review

1.4.2 Explore and Data colloection sushi dishes.

1.4.3 Data preprocessing

1.4.4 Develop an object detection model with the YOLOv4 algorithm.

1.4.5 Use file (*.weight) trained with the YOLOv4 algorithm as an application interface (API).

1.4.6 Test and evaluate research results.

1.4.7 Summary the result.

1.4.8 Compile and produce reseach paper.

1.4.9 Compile and produce thesis.

1.5 Outcomes

1.5.1 The speed of customer service in a conveyor belt sushi restaurant business

1.5.2 Use technology to reduce the risk of infection in epidemic situations from queuing at the counter.

1.6 Thesis Publication

Parts of the thesis had been published in the conference as following:

R. Maitriboriruks and Y. Limpiyakorn, “Object Detection for Classifying Sushi Dishes in Conveyor Belt Sushi Business”, in Proceedings of the 8th International Conference on Computer Technology Applications (ICCTA) , 2022 , Vienna , Austria

RANGRAK MAITRIBORIRUKS and PATCHARIYA PIYA-AROMRAT and YACHAI LIMPIYAKORN , “Smart Conveyor Belt Sushi Bill Payment with a Mobile Shot “, 2022, GuangZhou, China

Chapter 2
Related Theories and Literature Review

2.1 Related Theories

2.1.1 Object Detection

Object Detection [4] is a modern computer technology that deals with the application of Computer Vision and Image processing to detect various types of objects in images or videos by detecting objects with a number of researchers. The focus is on face detection and pedestrian detection along the road or various locations, and object detection can be divided into two main categories:

2.1.1.1 Multiple-Stage Object Detection

Multiple-Stage Object Detection [5] detects objects that use two or more models. Typically, the first model is used to separate regions and the second model is used to classify objects. (object classification) Multiple-Stage Object Detection is the detection of objects with high accuracy. The disadvantage is that it takes longer to detect other types of objects. Examples of models classified as Multiple-Stage Object Detection is Region Based Convolutional Neural Network (R-CNN) [6] invented by Ross Girshick et al. in 2014 and is said to be a CNN-based model (CNN) in The problem of object detection and segmentation is very good. Figure 3 shows the working of the R-CNN consisting of 3 modules:

- Region Proposal – Creating bounding box for object of interest
- Feature Extractor - Serves to extract features from each region of the object of interest using Deep CNN.
- Classifier - Use to classify the object of interest as an object class by using the SVM classifier.

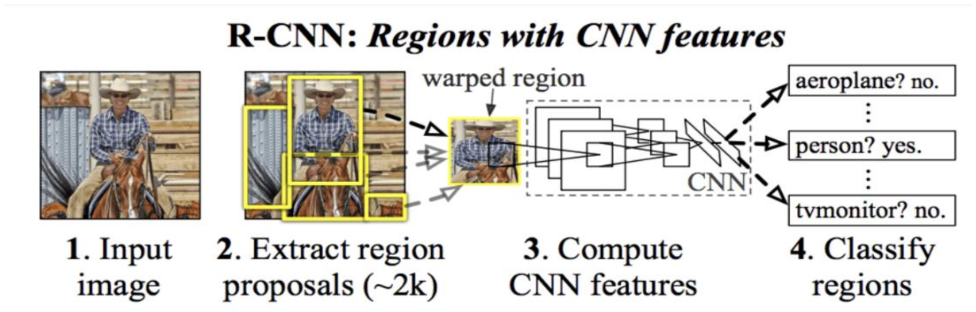


Figure 3: R-CNN workflow [6]

2.1.1.2 Single-Stage Object Detection

Single-State Object Detection [6] is an object detection that uses only one model to detect all classes of objects in an image or video. The advantage is that Single-State Object Detection takes less training time than multiple- stage object detection and suitable for use with mobile devices and also has high accuracy.An example of a Single-Stage Object Detection is YOLO. It is a model developed by Joseph Redmon et al. Figure 4 shows the YOLO workflow starting from the S x S grid division of the image. Each grid is responsible for predicting the bounding box if the center of the bounding box lies within each grid. Each grid predicts the bounding box with respect to x ,y, width, height, sentiment. (confidence) and use Non Maximal Suppression to find bounding boxes with the highest Intersection Over Union (IoU) value, reducing the problem of bounding boxes overlapping in multiple layers

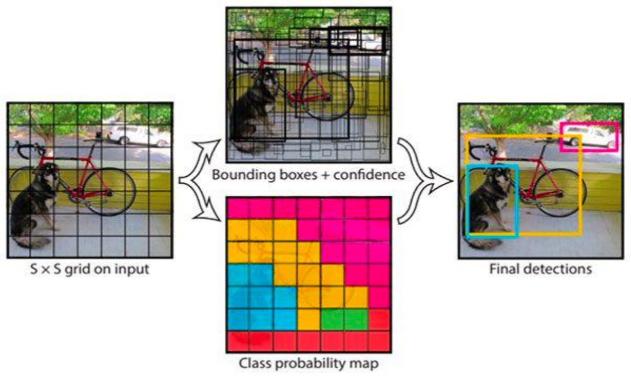


Figure 4: YOLO workflow [6]

2.1.2 Spring Boot REST API

Spring Boot [7] developed by Pivotal, is an open-source framework that uses Java as its primary language. Spring Boot uses the concept of Microservices, which clearly separates each function into sub-services by Spring. Boot adopts the concept of REST, an architectural style that takes advantage of Web protocol technologies to create Web services to create application interfaces. API in order to make the client (Client) able to communicate with the server (Server)

2.1.3 OpenCV

OpenCV [8] developed by Intel, is a library with the goal of real-time computer rendering. for use in the development of open-source software systems OpenCV is used to perform visualizations in Machine Learning or Artificial Intelligence in object recognition or Face Recognition problems. Developed with C++ and also supports Python, Java and MathLab, the OpenCV is a cross-platform library. It is cross-platform and freely available under the Berkeley Source Distribution (BSD) open-source software license. Detect objects of interest.

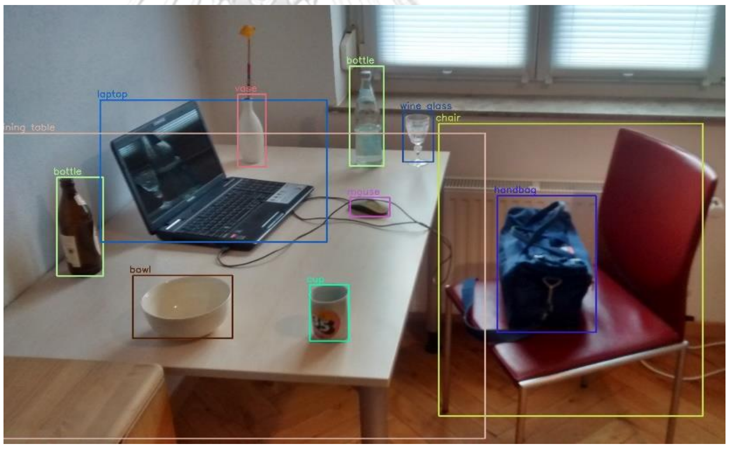


Figure 5: Object detection using OpenCV trained by YOLO [4]

2.2 Literature Review

2.2.1 Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition [9]

Jeong-ah Kim et al. conducted research on comparing three neural network models, 1. Faster-RCNN, 2.YOLOv4, and 3.Single-Shot Detector (SSD), with the Automobile dataset, which were more efficient and have the most accurate in real-time object detection. The research categorizes vehicles into 6 categories: 1.car 2.mini_van 3.big_van 4.mini_truck 5.truck and 6.compact. Before taking the dataset for training, a Region of Interest (ROI) was done to label the objects of interest before training with the model as shown in Figure 6. The SSD model uses mobilenet v1, the Faster-RCNN model uses Inception v2 and YOLOv4 is all used by default. Comparisons of the results are summarized in Table 1.

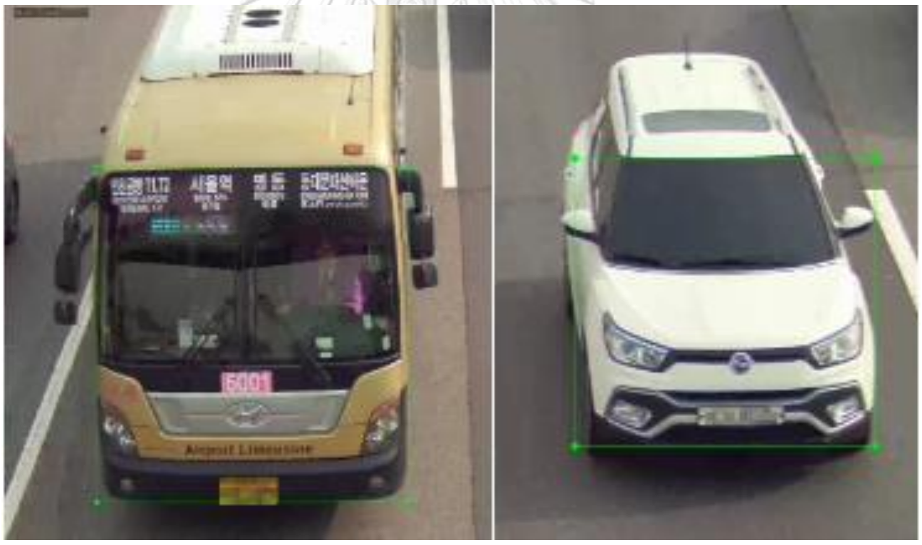


Figure 6: ROI before training with the model [9]

Table 1 shows that Faster-RCNN is a model with two state object detection: regional proposal and classification. Low accuracy was obtained with real-time datasets of type F1score=0.90 , Precision=0.86 , Recall=0.94 , and mAP=93.40 makes Faster-RCNN unsuitable for use. The SSD has the fastest speed compared to Faster-RCNN and YOLOv4 because it uses a mobilenet light model, but the accuracy is the lowest compared to Faster-RCNN and YOLOv4, with F1score=0.88 , Precision=0.90 , Recall=0.87 and mAP

= 90.56, making SSD unsuitable for adoption. YOLOv4 is a very accurate model that predicts vehicles without fail in real time. However, the model training speed is slower than SSD but faster than Faster-RCNN: F1score=0.96 , Precision=0.93 , Recall. =0.98 and mAP=98.19, which is the highest achievement compared to Faster-RCNN and SSD, thus YOLOv4 has the best predictive performance compared to Faster-RCNN and SSD.

Table 1: Comparison of model performance: Faster-RCNN, SSD, YOLOv4 [9]

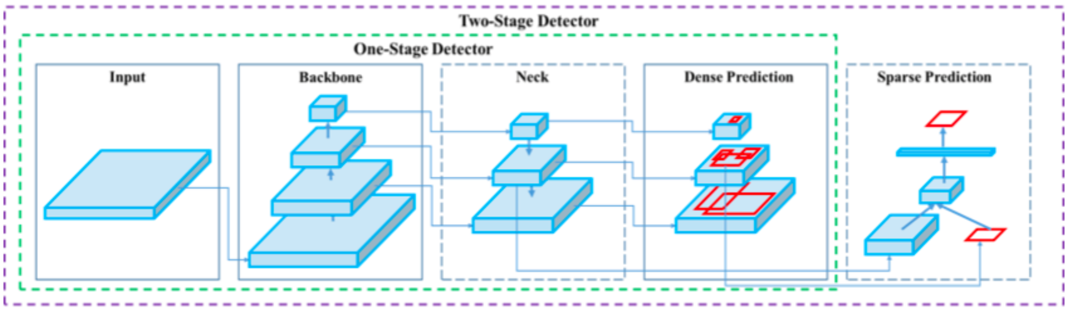
	F1score	*Precision*	*Recall*	*mAP*	*Time*
Faster-RCNN	***0.90***	***0.86***	***0.94***	***93.40***	***Ranking 3***
SSD	***0.88***	***0.90***	***0.87***	***90.56***	***Ranking 1***
YOLOv4	***0.93***	***0.98***	***0.98***	***98.19***	***Ranking 2***

2.2.2 YOLOv4: Optimal Speed and Accuracy of Object Detection [10]

Alexey Bochkovskiy et al. implemented YOLOv4 to overcome the drawbacks of Convolutional Neural Networks that suffer the detection of overlapping objects, real-time detection, and high GPU consumption. Figure 4 illustrates the four major learning steps consisting of:

- Input - the training images
- Backbone - CSPDarknet53 is chosen as the pretrained model for object recognition. Compared to CSPResNet50 and EfficientNet-B3, CSPDarknet53 provides higher accuracy rate and less training time.
- Neck - combine the features in conv net as preparation before moving to the next step. The PANet model was selected for feature integration. The block following CSPDarknet53 was implemented with the SPP model to augment the receptive field and separate important features from the backbone.
- Head - apply Head of YOLOv3 to predict the bounding box used for classification and regression. The format of the enclosing framework is divided into 4 characters: width, height, center, and label prediction score.

As a result, using YOLOv4 achieved 43.5% speed (AP) from the real-time MS COCO dataset of 65 Frame per second (FPS) on the Tesla V100 GPU. When comparing YOLOv4 with YOLOv3, the Average Precision (AP) increased by 10% and FPS increased by 12%. The benchmark of YOLOv4 against other single stage object detection models using the same dataset MS COCO is depicted in Figure 8. The graph shows that YOLOv4 achieved the highest average Precision /frames per second, indicating that YOLOv4 had the highest accuracy despite the increased frames per second that could be applied. YOLOv4, therefore, works well for object detection of real-life photography [11].



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 7: Steps of object detection of YOLOv4 [10]

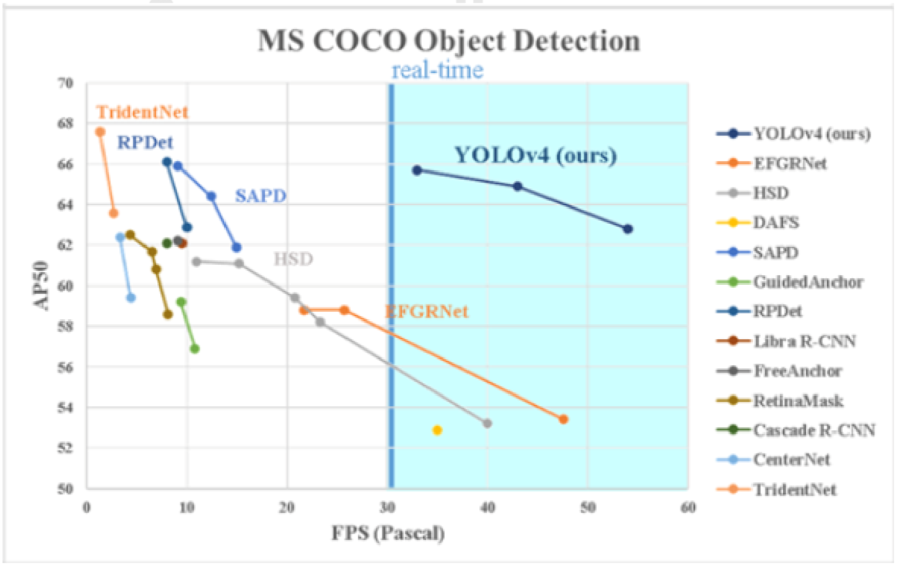


Figure 8: Yolov4 BenchMark [11]

2.2.3 Automatic Counting Shrimp Larvae Based You Only Look Once (YOLO) [12]

Siska Armalivia et al. conducted research on using YOLOv3 3 model to help shrimp embryo count as shown in Figure 9 to help shrimp farmers to determine the number of shrimp embryos. for calculating profit or loss The research process is as follows.

- Data Collection - A total of 355 images of shrimp embryos were collected by mobile phones taken at the Takalar Brackish Water Aquaculture Fisheries, Center Takalar Regency and South Sulawesi.
- Annotation Image - Bring the image dataset to be converted into a file format Label image using YOLO mark tools. By creating a Label image will set a frame to the object of interest and the resulting value will be in the file format (*.txt)
- Training the Model - Before training the model with the dataset, edit the .cfg file by setting the learning rate=0.001, batches=64, iteration=4000. The dataset was divided into 325 training data and 30 testing data. After changing the values in the cfg file and dividing the dataset, the model was trained using the Graphics Processing Unit (GPU).

The research result achieved F1score=94.38%, Recall=93.51%, Precision=95.26%, mAP@0.5=96.83% and average of Accuracy=76.48%. Note that the Precision is very high and when the larger the number of images will make the value more accurate.

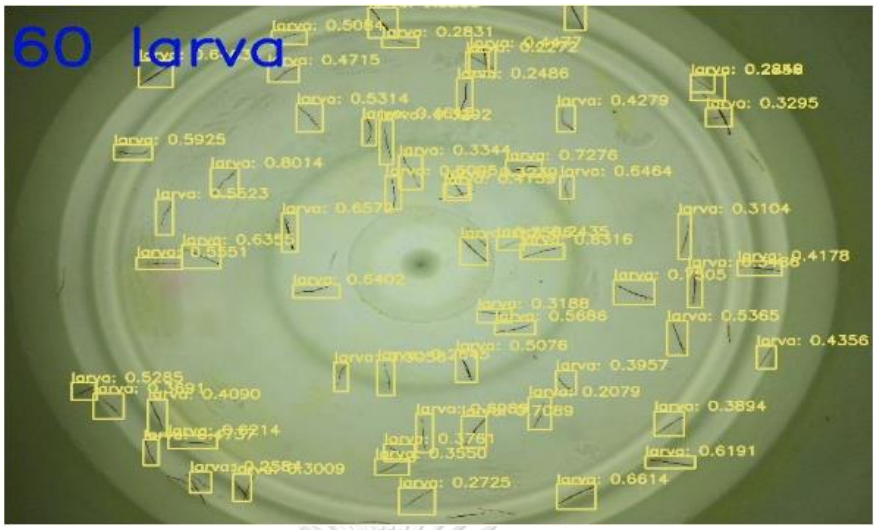


Figure 9: Example of an embryo counting image [12].

2.2.4 Sushi Dish : Object detection and classification from real images [13]

Yeongjin Oh et al. 2017 conducted research on the object detection and classification of sushi plates from real images. The researchers implemented Ellipse Detection method to detect sushi plates as shown in Figure 10. The Convolutional Neural Network model was trained to classify color-coded plates as shown in Figure 11. The model achieved ellipse detection precision 85%, recall 96%, and classification accuracy 92%

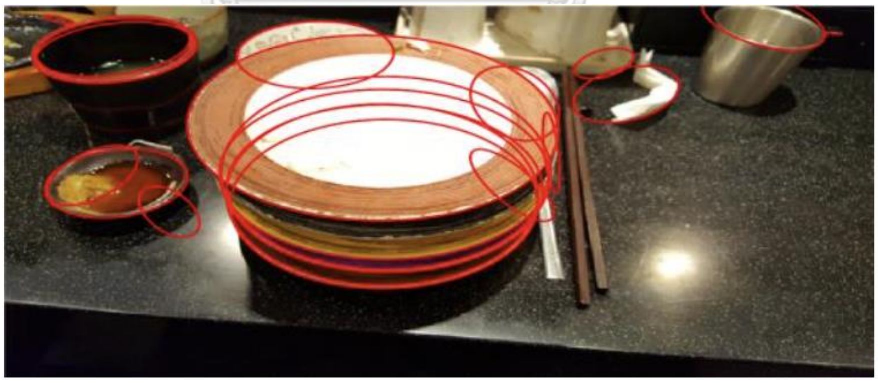


Figure 10: Ellipse Detection method to detect sushi plates [13]

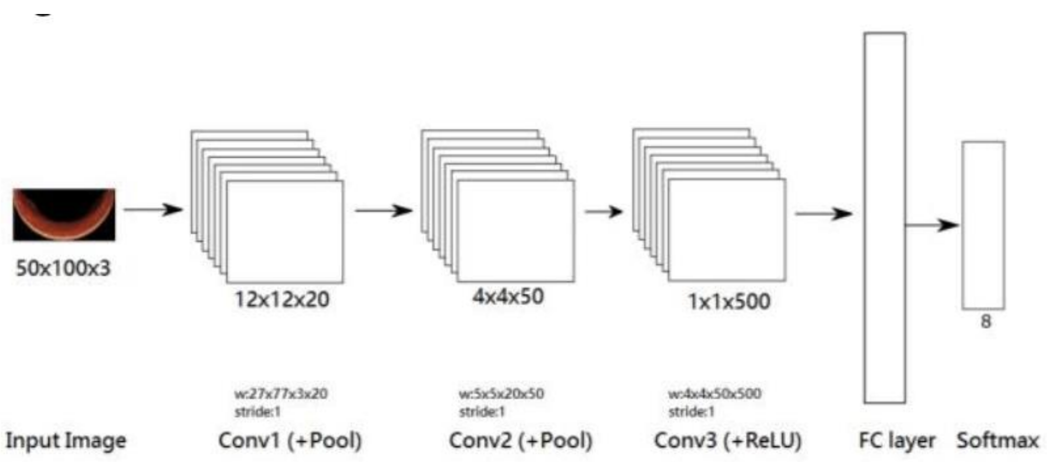


Figure 11: Convolutional Neural Network for classify sushi plates [13]

Chapter 3
Proposed Methodology

This chapter mainly describes the construction of an object detection model implemented with YOLOv4 to facilitate the automated count of distinct colored sushi dishes. Figure 12 illustrates the architectural design of the proposed smart billing system for a conveyor belt sushi restaurant. The system structure can be divided into three main components. The frontend is developed with Flutter to build single codebase for UIs. The intelligent component of sushi plate count or the object detection model is constructed using YOLOv4. The backend is designed with Microservices architecture. Orchestration of YOLOv4, OpenCV and the Spring Boot REST API will create an API (Application Programming Interface) offering a service to other pieces of software. For example, when the API has finished calculating the cost of food, it will store the food cost for billing and the customer's information for membership benefits in the database. The Spring Boot REST API is selected as it can support a large number of incoming requests by making the API itself as a Microservice. While OpenCV [8] is a huge Opensource Computer Vision library that contributes to image processing and performing computer vision tasks. The library is cross-platform and mainly aimed at real-time computer vision.

3.1 YOLOv4 Model

According to related research, the findings of using the YOLOv4 approach to train an image dataset revealed high accuracy and the capacity to discriminate overlapping items [12]. As a result, this approach is appropriate for the sushi plate's visual qualities. Sushi platters are naturally sorted into column before being counted. As a result, the researcher chose the YOLOv4 approach as a training method for the image dataset, dividing the dataset into 80% for training and 30% for testing. Setting the pre-trained weight to yolov4.conv.137, divided into 4 main steps as described following:

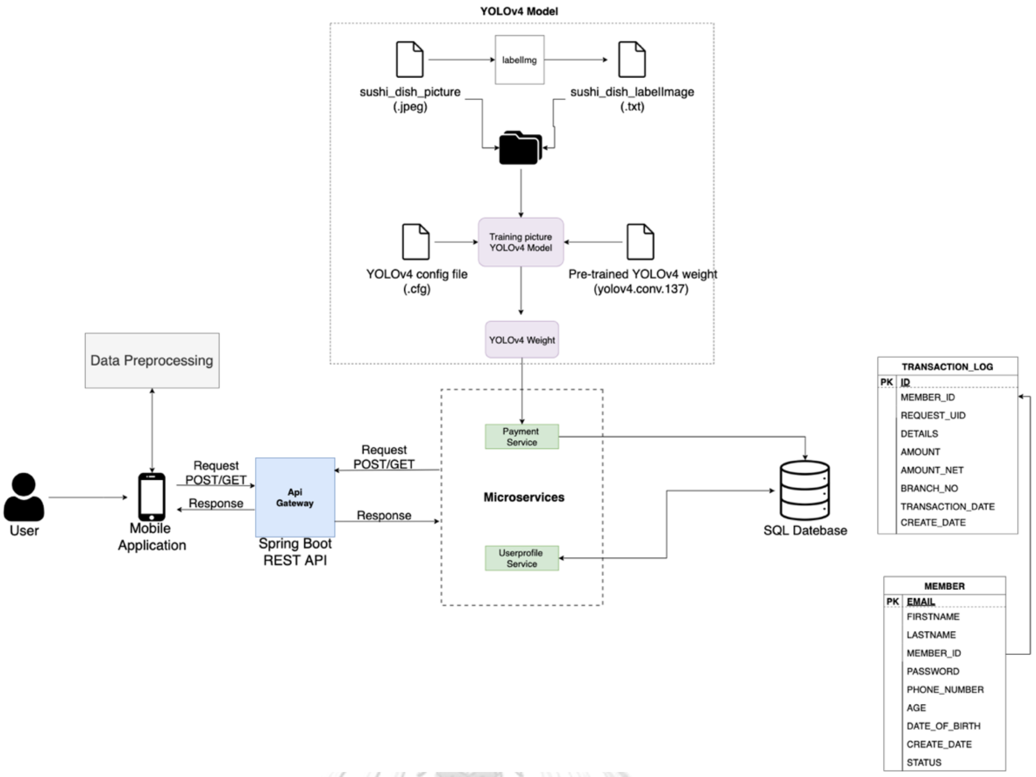


Figure 12: Architecture Design of Integrated Research System

3.1.1 Image collection of how the sushi plates is arranged in layers with different color-coded plates from the iPhone 12 in JPEG (.jpeg) file format. As indicated in Figure 13, the photographs collected are a plate of sushi arranged according to the nature of the restaurant's clients. Sushi plates have a clear habit of being sorted in a variety of ways. It can be arranged both in color order and out of color order. It could also include non-sushi plates. When there are a lot of plates, it is usually organized in more than one column. Various objects, like as sauce dishes, free cups, chopsticks, and food trash, are frequently placed on top of sushi platters.

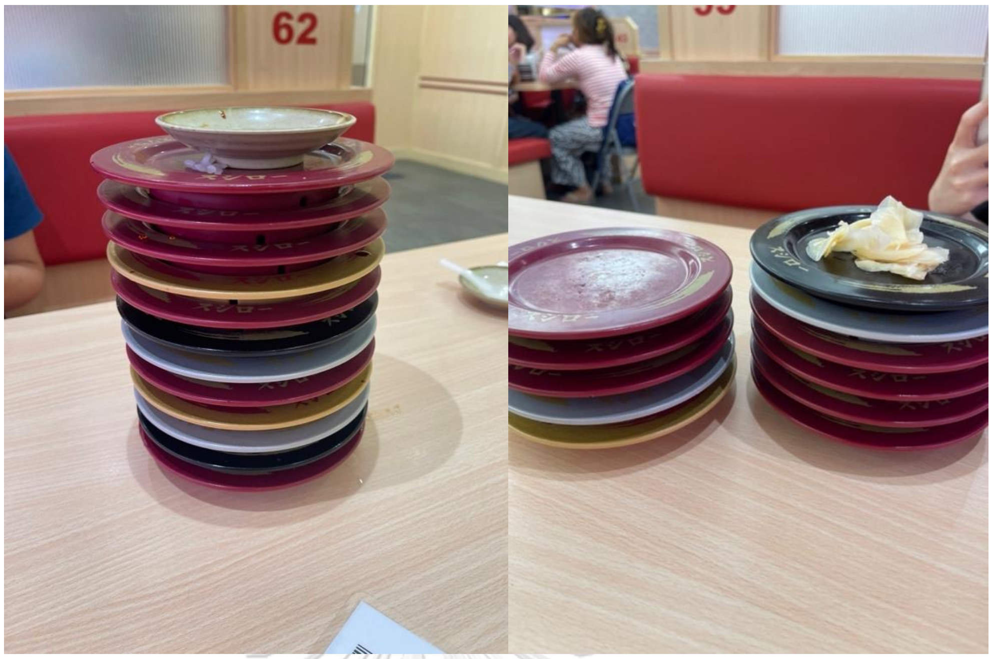


Figure 13: example of sushi plates

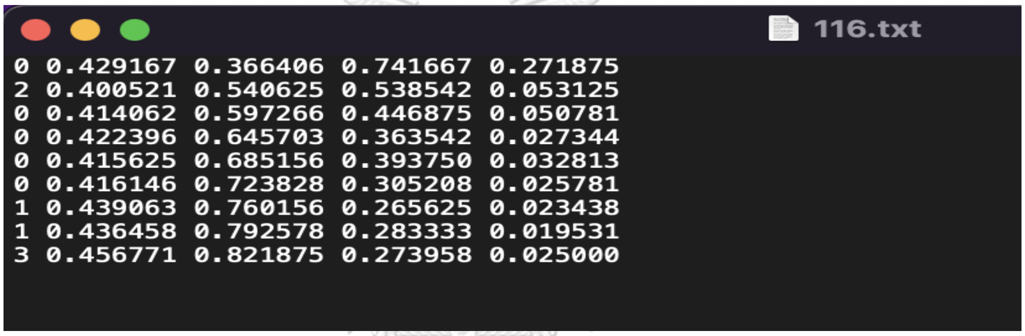
3.1.2 Label Image

The initial step was to collect photos from a dataset and label it, which was called Annotation Image. Label image in order to identify the position of the objects of interest, before using the YOLOv4 approach. The label image's value is determined by five factors as Figure 14:

- Category Number is the class of the object of interest consisting 7 classes: 0) “40baht” represents the red sushi plate, 1) “60baht” represents the silver sushi plate, 2) “80baht” represents the golden sushi plate, 3) “120baht” represents the black sushi plate, 4) “free-water” represents the free water cup, 5) “free-dish” represents the free sauce plate, and 6) “not-sushi-dish” represents other kinds of plates used for special orders. Note that in this work, the class label indicates the product pricing rather than the type of sushi plate.
- *Bounding Box Left X* denotes the initial x-axis position of the object of interest at the upper left corner.

- *Bounding Box Top Y* denotes the initial y-axis position of the object of interest at the upper left corner.
- *Bounding Box Right X* denotes the starting x-axis of the object of interest at the lower right corner.
- *Bounding Box Bottom Y* denotes the starting y-axis of the object of interest at the lower right corner.

Labelimg [14], an open-source software that allows you to configure an image in the form of a label image that looks like a text (.txt) file. In figure 4, Labelimg can be used to create a label image.



```
116.txt
0 0.429167 0.366406 0.741667 0.271875
2 0.400521 0.540625 0.538542 0.053125
0 0.414062 0.597266 0.446875 0.050781
0 0.422396 0.645703 0.363542 0.027344
0 0.415625 0.685156 0.393750 0.032813
0 0.416146 0.723828 0.305208 0.025781
1 0.439063 0.760156 0.265625 0.023438
1 0.436458 0.792578 0.283333 0.019531
3 0.456771 0.821875 0.273958 0.025000
```

Figure 14: Label images format

3.1.3 Config the parameter in YOLOv4-custom.cfg

change the values in the file (*.cfg) as follows: batch at 64, subdivisions at 16, width at 416, height at 416, max_batches at 14000 (all value from class are multiplied by 2000), steps at 11200,12600(the first value is 80% of the max_batches, and the second value is 90% of the max_batches) filters at 33 (values are based on (class+5)*3) claesses at 7 (values from the number of classes of objects of interest), all of these numbers are based on the inventor of YOLOv4 [15]

3.1.4 Pre-Trained Weight YOLOv4

Download the yolov4.conv.137 file, which is the YOLOv4 Pre-Training Weight file used to train datasets that are not in the MS COCO dataset. The research datasets are conveyor belt sushi dishes.

3.2 API Gateway

The Spring Boot REST API was used in this phase to help create the application interface so that data could be exchanged between the frontend and backend. According to related research studies, the majority of them were built in Python. However, in this study, Spring Boot REST API was chosen because of JAVA unique property of being able to write Microservices better than Python [15]. This approach is the best way to manage in parts when using the Microservice architecture. Managing future upgraded versions is undeniably convenient. Another advantage of using the Spring Boot REST API is having a database ecosystem that is simple to connect to the database and suitable of making application interfaces.

3.3 Microservices

This section is divided into 2 main Microservices:

3.3.1 Payment Service

The payment service's main function is to process the weight value received from YOLOv4 training and apply it to the application interface. It can then determine the total payable amount to the customer and, finally, record the customer's daily data in the database. The Request body is being rolled out as a photo base64 and customer data receiver. The calculated amount, as well as the number of sushi color-coded plates that clients have ordered, are returned to the Response.

3.3.2 **Userprofile Service**

The User Profile Service's main function is to store and retrieve customer information that has been subscribed to the database.

3.4 Database

The database was built using Structured Query Language (SQL). To further explain the advantages of SQL, it is straightforward to work when table features are in a fixed layout format. TRANSACTION_LOG and MEMBER_LOG are the only two variables in the database. To be clear, TRANSACTION_LOG records the details of consumers who visit the conveyor belt sushi restaurant on a daily basis. MEMBER_LOG stores information on customers who have signed up to be restaurant members.

3.5 Model Evaluation

Use the confusion matrix metrics including recall, precision, f1-score and mean average precision for accuracy. For the accuracy of sushi plates calculations will test by call the API with postman

Chapter 4
Evaluation and Result

4.1 Object Detection Model

In this work, a unified, real-time object detector was constructed with YOLOv4 to classify the type of sushi plates in conveyor belt sushi business. The performance of YOLOv4 models tends to achieve high accuracy and the capacity to discriminate overlapping items. The image dataset used for model construction is collected from a Japanese revolving sushi restaurant franchise in Bangkok, Thailand.

4.1.1 **Data Collection**

Prior to count the number of consumed dishes, the plates are typically arranged in layers with different colors. A set of 600 photographs was taken by a smartphone— iPhone 12 pro max. Figure 15 illustrates example consumed dishes gathered from the Japanese's most famous conveyor belt sushi franchise restaurant in Bangkok, Thailand. Figure 11a) portrays the stacks overlap each other. Figure 11b) contains a noise of a sauce plate with a spoon at the top of stack. Figure 11c) displays a close-up shot. Only half-side of stack is captured in Figure 11d). Figure 11e) contains noises of chopsticks at the top of stack. Example of the high stack is depicted in Figure 11f). Figures 11g), 11h), and 11i) present the cases where the top plate is upside-down, and the same color as the beneath; but not sushi plate; and different color from the beneath, respectively.

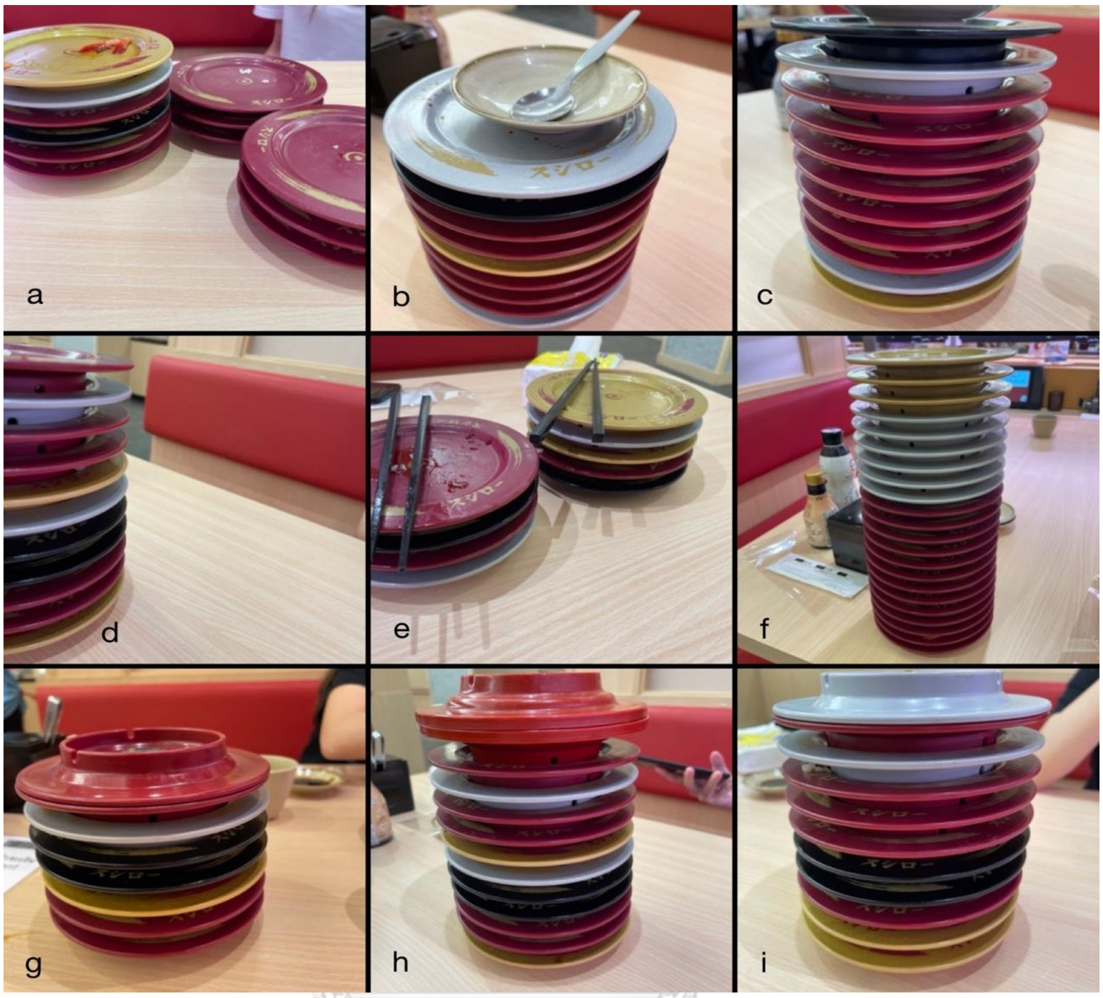
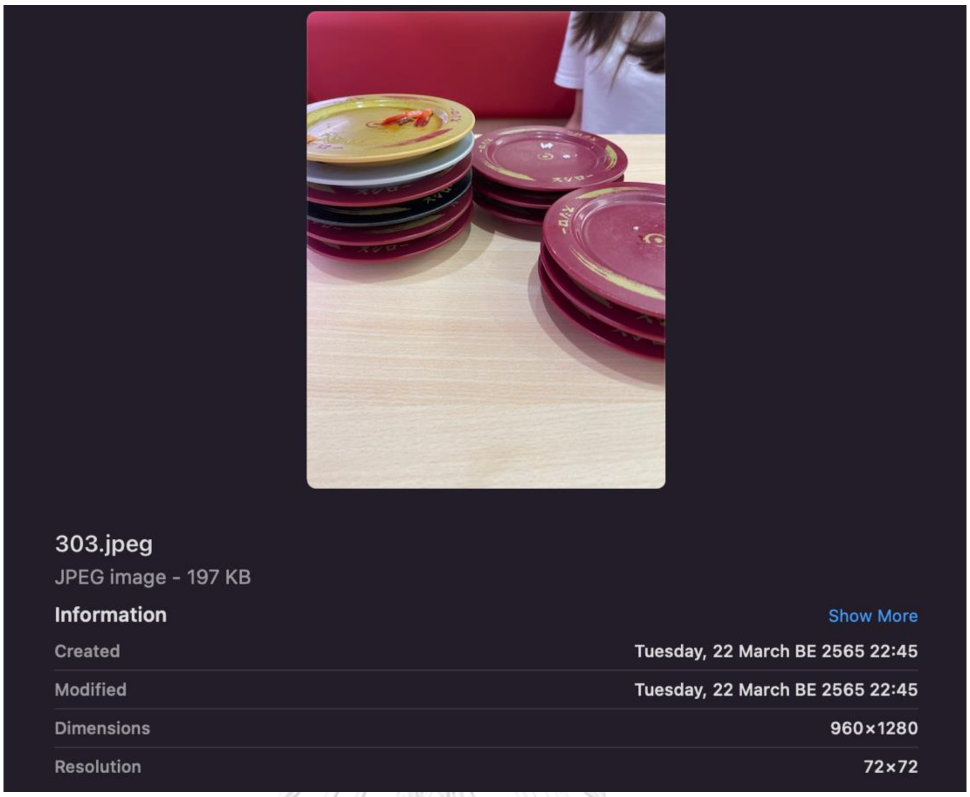


Figure 15: Example training images.

4.1.2 Data Preprocessing

Use Preview, a program that has the ability to image viewer and PDF viewer of the macOS operating system. In addition to viewing and printing digital images and Portable Document Format files, it can also edit these media types [16] to transform the image format from (*.HEIC) to (*.JPEG). Next, the dimensions of images were converted to 960x1280 following the input format of YOLOv4 as shown in Figure16.

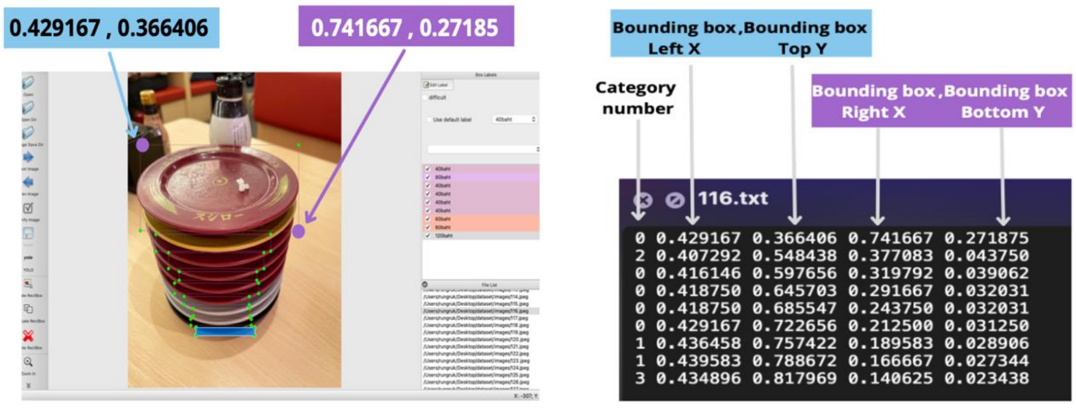

303.jpeg
JPEG image - 197 KB
Information
Show More

Created	Tuesday, 22 March BE 2565 22:45
Modified	Tuesday, 22 March BE 2565 22:45
Dimensions	960×1280
Resolution	72×72

Figure 16: Example after data preprocessing images.

4.1.3 Image annotation

Manual image annotation was performed to define the regions of interest in an image and label a textual description of those regions. Labelimg, an open-source software, is used as a tool to facilitate the task of configuring an image in the form of a label image that looks like a text (.txt) file, as shown in Figure 17. The directory after making the label image will look like in Figure 18. there is a file (*.JPEG) converted in the data preprocessing step along with a file (.txt) obtained from the Label image. After that compression directory in a format (.zip) and upload it to Google drive for use in the training model process.



```
0 0.429167 0.366406 0.741667 0.271875
2 0.407292 0.548438 0.377083 0.043750
0 0.416146 0.597656 0.319792 0.039062
0 0.418750 0.645703 0.291667 0.032031
0 0.418750 0.685547 0.243750 0.032031
0 0.429167 0.722656 0.212500 0.031250
1 0.436458 0.757422 0.189583 0.028906
1 0.439583 0.788672 0.166667 0.027344
3 0.434896 0.817969 0.140625 0.023438
```

Figure 17: (a) screen capture of Labelimg showing bounding box. (b) label image format(.txt).

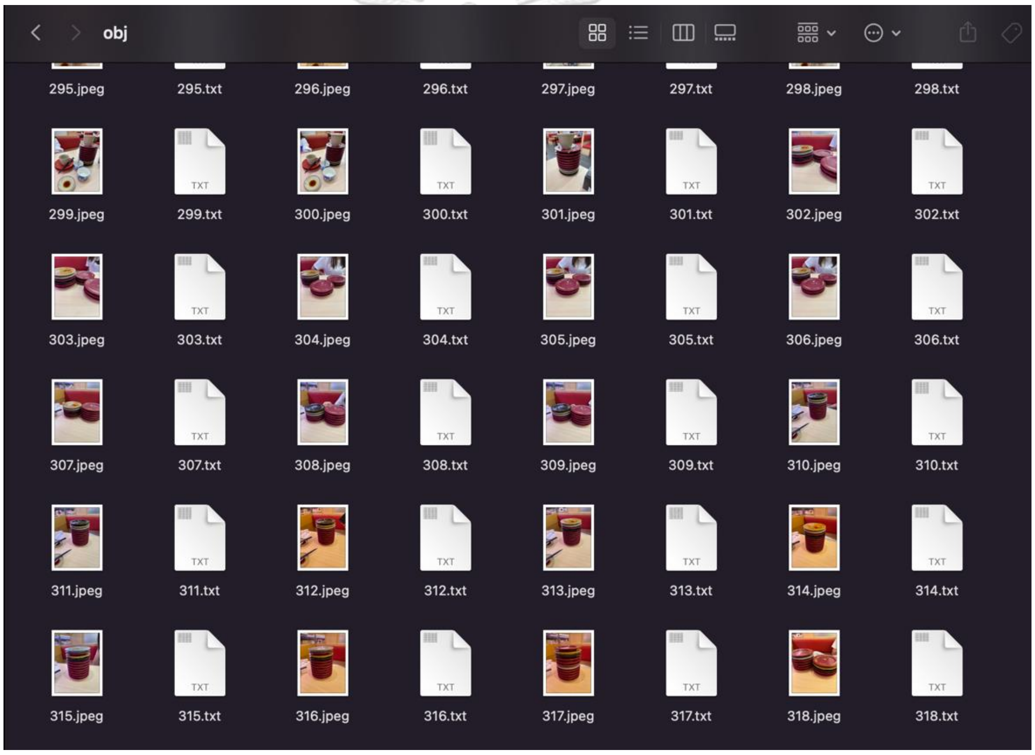
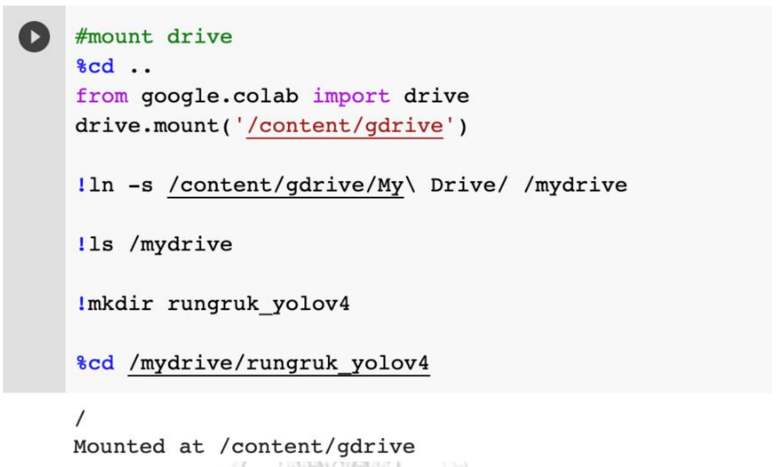


Figure 18: screen capture directory after labeling images

4.1.4 Model Training

The model was trained on the Google cloud (colab pro) with the specification of GPU = T4, RAM size = 32GB. Training the model on cloud is easy to maintain and the system is stable.

Use Google drive to access the dataset (*.zip) made in Data preprocessing and Image Annotation steps and to store the file (*.weight) obtained from the YOLOv4 train model. The advantage of storing the file (*.weight) is when training model and then disconnected or lose session, no need to retrain the model from the beginning but can bring the last saved file (*.weight) to run where the file (*.weight) will be saved every 100 iteration [17] In google drive need to mount drive to retrieve data in google colab Use the command drive.mount('/content/drive') under the google.colab library and !mkdir ~dir to create a directory to store the YOLOv4 library and files (*.weight) as shown in Figure19.

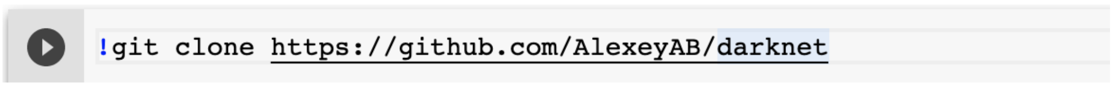


```
#mount drive
%cd ..
from google.colab import drive
drive.mount('/content/gdrive')

!ln -s /content/gdrive/My\ Drive/ /mydrive

!ls /mydrive

!mkdir rungruk_yolov4

%cd /mydrive/rungruk_yolov4
```

```
/
Mounted at /content/gdrive
```

Figure 19: Code for connecting google drive with google colab

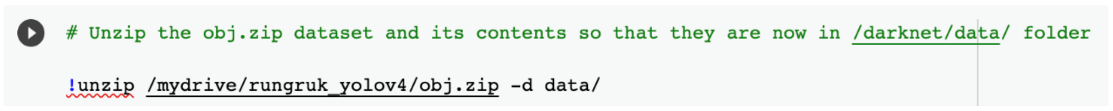
Download the YOLOv4 library from github and store it in the directory ~/mydrive/rungruk_yolov4 using the !git clone https://github.com/AlexeyAB/darknet command as shown in Figure 20.



```
!git clone https://github.com/AlexeyAB/darknet
```

Figure 20: Code for download YOLOv4 library

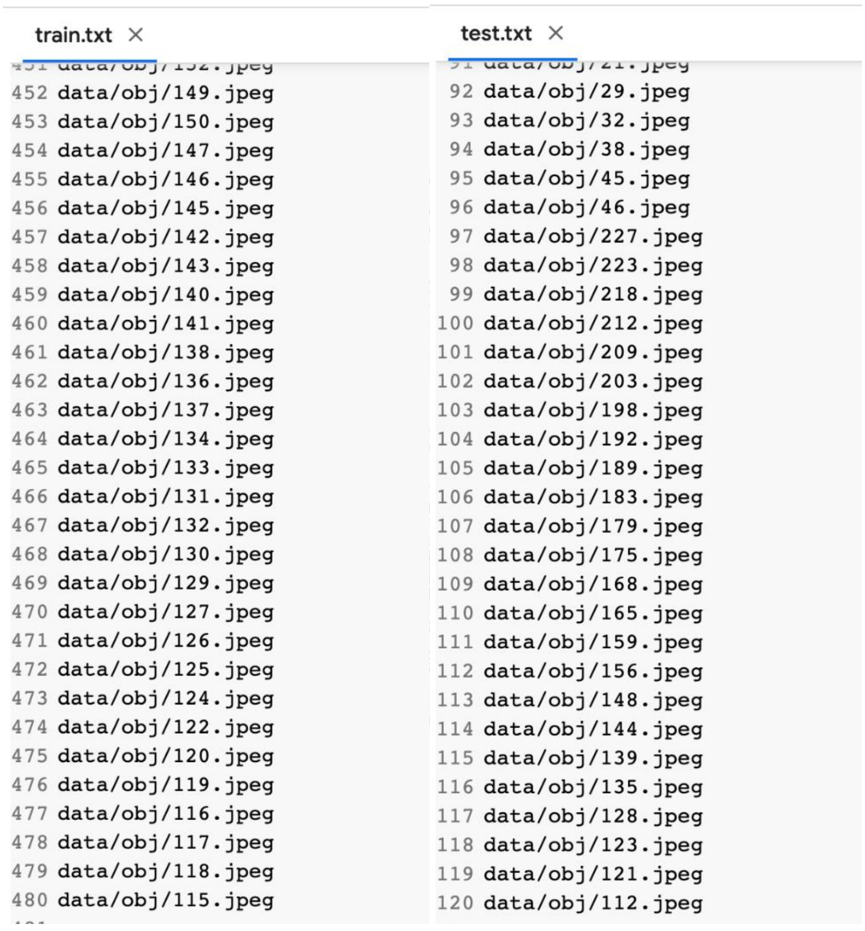
Unzip the dataset on Google drive to directory ~/rungruk_yolov4/darknet/data using the command !unzip /mydrive/rungruk_yolov4/obj.zip -d data/ as shown in Figure 21.



```
# Unzip the obj.zip dataset and its contents so that they are now in /darknet/data/ folder

!unzip /mydrive/rungruk_yolov4/obj.zip -d data/
```

Figure 21: Code for unzip dataset to directory ~/rungruk_yolov4/darknet/data

Once the unzip dataset is complete, The dataset of 600 images was divided into train and test sets with the ratio 80:20, i.e., 480 images of training data will be written to train.txt and 120 images of testing data will be written to. File test.txt as shown in Figure 22. Figure 23 is a code to randomly split dataset in ratio 80:20 and write to file train.txt and test.txt.



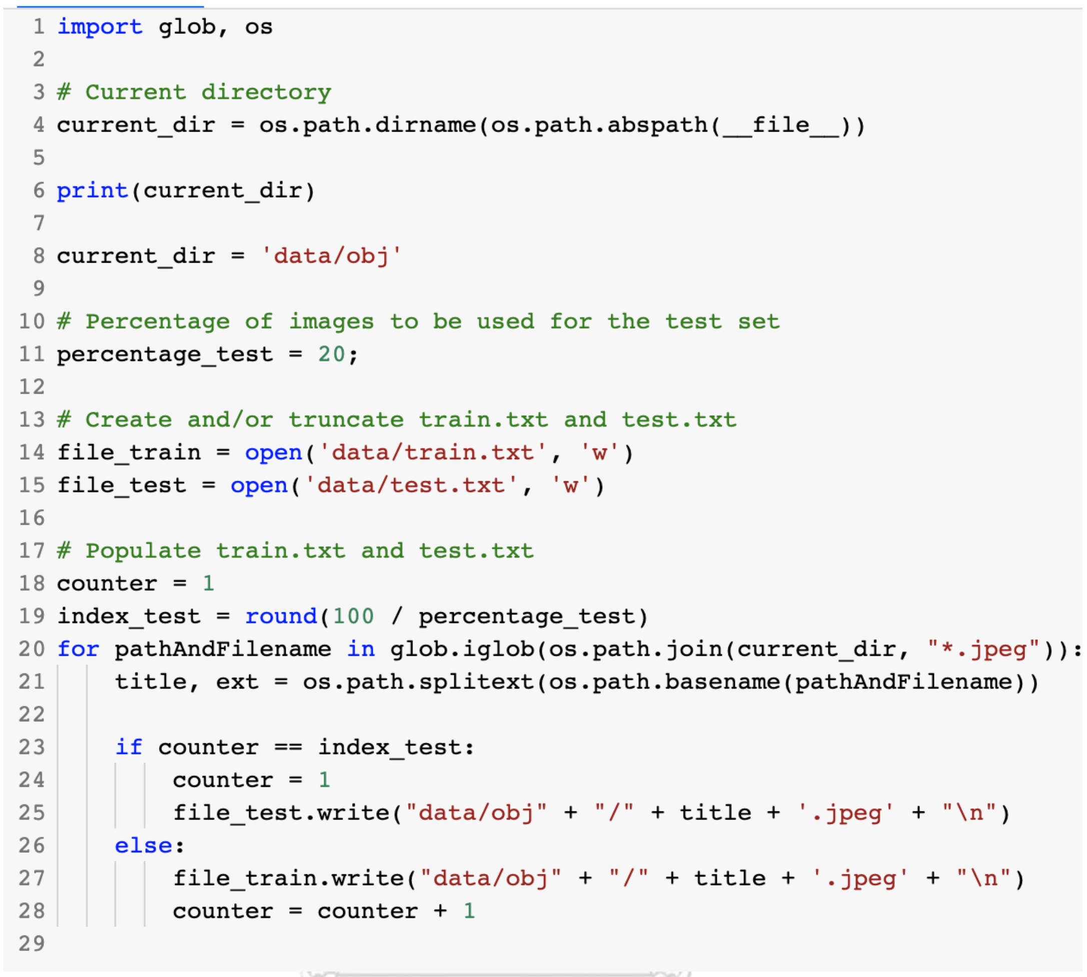
train.txt ×

```
data/obj/149.jpeg
data/obj/150.jpeg
data/obj/147.jpeg
data/obj/146.jpeg
data/obj/145.jpeg
data/obj/142.jpeg
data/obj/143.jpeg
data/obj/140.jpeg
data/obj/141.jpeg
data/obj/138.jpeg
data/obj/136.jpeg
data/obj/137.jpeg
data/obj/134.jpeg
data/obj/133.jpeg
data/obj/131.jpeg
data/obj/132.jpeg
data/obj/130.jpeg
data/obj/129.jpeg
data/obj/127.jpeg
data/obj/126.jpeg
data/obj/125.jpeg
data/obj/124.jpeg
data/obj/122.jpeg
data/obj/120.jpeg
data/obj/119.jpeg
data/obj/116.jpeg
data/obj/117.jpeg
data/obj/118.jpeg
data/obj/115.jpeg
```

test.txt ×

```
data/obj/29.jpeg
data/obj/32.jpeg
data/obj/38.jpeg
data/obj/45.jpeg
data/obj/46.jpeg
data/obj/227.jpeg
data/obj/223.jpeg
data/obj/218.jpeg
data/obj/212.jpeg
data/obj/209.jpeg
data/obj/203.jpeg
data/obj/198.jpeg
data/obj/192.jpeg
data/obj/189.jpeg
data/obj/183.jpeg
data/obj/179.jpeg
data/obj/175.jpeg
data/obj/168.jpeg
data/obj/165.jpeg
data/obj/159.jpeg
data/obj/156.jpeg
data/obj/148.jpeg
data/obj/144.jpeg
data/obj/139.jpeg
data/obj/135.jpeg
data/obj/128.jpeg
data/obj/123.jpeg
data/obj/121.jpeg
data/obj/112.jpeg
```

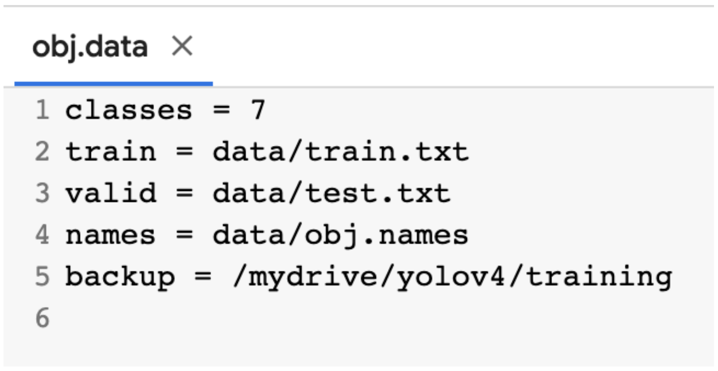
Figure 22: Screen capture Train.txt and Test.txt



```python
import glob, os

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)

current_dir = 'data/obj'

# Percentage of images to be used for the test set
percentage_test = 20;

# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpeg")):
    title, ext = os.path.splitext(os.path.basename(pathAndFilename))

    if counter == index_test:
        counter = 1
        file_test.write("data/obj" + "/" + title + '.jpeg' + "\n")
    else:
        file_train.write("data/obj" + "/" + title + '.jpeg' + "\n")
        counter = counter + 1

```

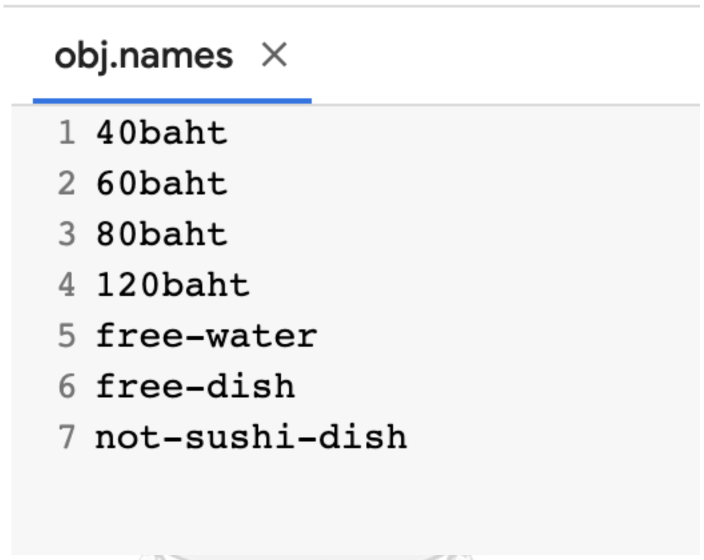
Figure 23: Python code for split data

Create obj.data and obj.names files in directory ~/rungruk_yolov4/darknet/data where obj.data is a config path file that tells each parameter to read from that path, by setting classes = 7, train = data/train.txt, valid = data/test.txt, names = data/obj.names, backup = /mydrive/yolov4/training as shown in Figure24, and The obj.names file tells the class of the objection of interest consisting of how many classes and what is the name of each class as shown in Figure25.


obj.data ×

```
classes = 7
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov4/training
```

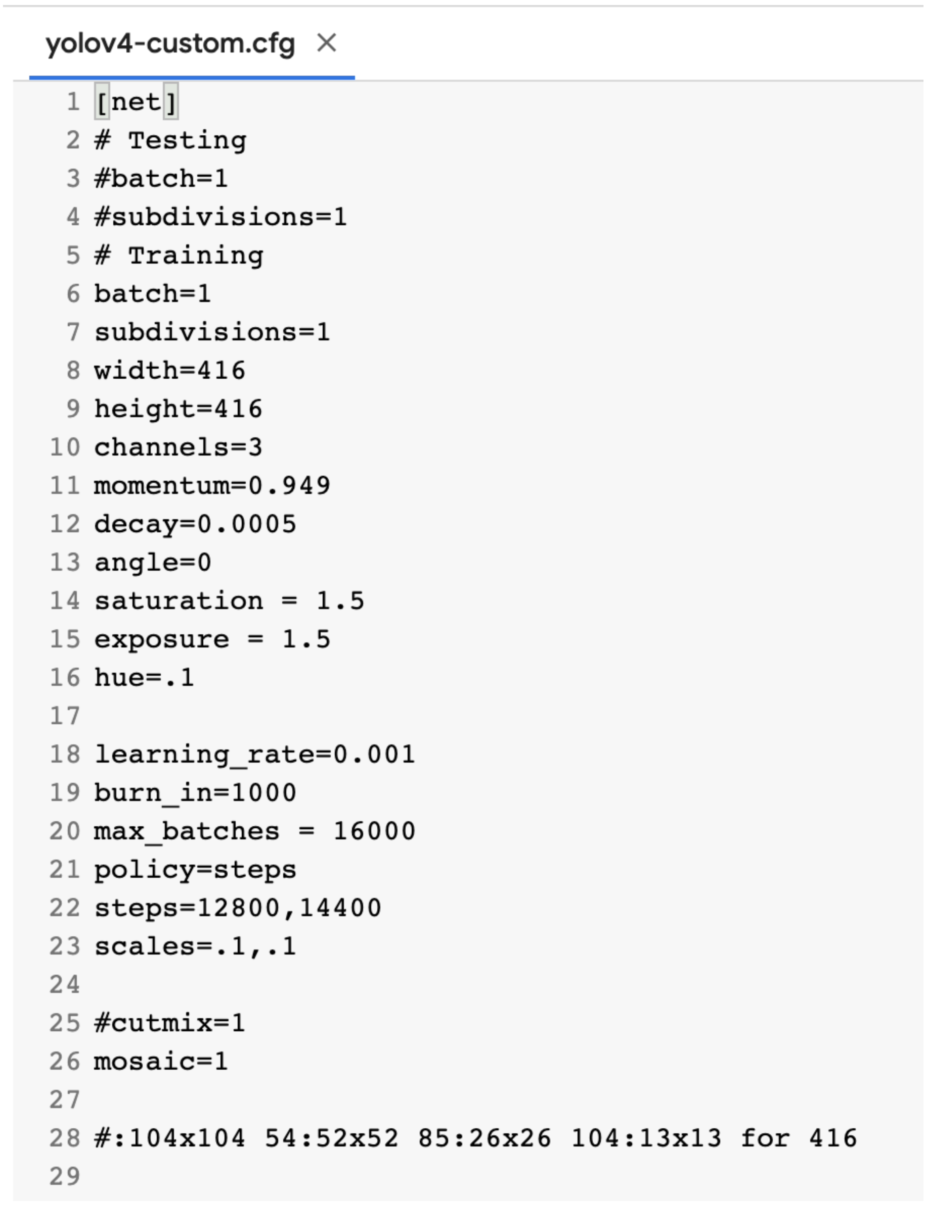
Figure 24: Capture screen of obj.data


obj.names ×

```
40baht
60baht
80baht
120baht
free-water
free-dish
not-sushi-dish
```

Figure 25: Capture screen of obj.names

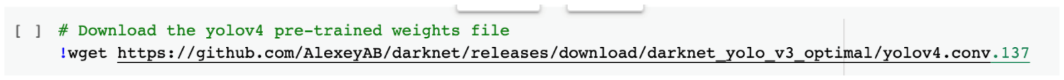
Modify the values in the file (yolov4-custom.cfg) in directory ~/rungruk_yolov4/darknet/cfg as follows: batch at 64, subdivisions at 16, width at 416, height at 416, max_batches at 16000 , steps at 12800,14400 (the first value is 80% of the max_batches, and the second value is 90% of the max_batches), filters at 36 (values are based on (class+5)*3), classes at 7 (values from the number of classes of objects of interest) as shown in Figure26. Some parameter settings are the defaults suggested by the inventor of YOLOv4 [18]. Each variable in the config file can be described as follows. batch is a number of samples images which will be processed in one batch , subdivisions is a number of mini_batches in one batch, size mini_batch = batch/subdivisions, so GPU processes mini_batch samples at once, and the weights will be updated for batch samples (1 iteration processes batch images) , width and height is a network size of width and height, so every image will be resized to the network size during Training and Detection, steps is an Adjust of the learning rate after 500 and 1000 batches , max_batches is the training will be processed for this number of iterations and filter is a number of kernel-filters.



```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=1
subdivisions=1
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 16000
policy=steps
steps=12800,14400
scales=.1,.1

#cutmix=1
mosaic=1

#:104x104 54:52x52 85:26x26 104:13x13 for 416

```

Figure 26: yolov4-custom.cfg

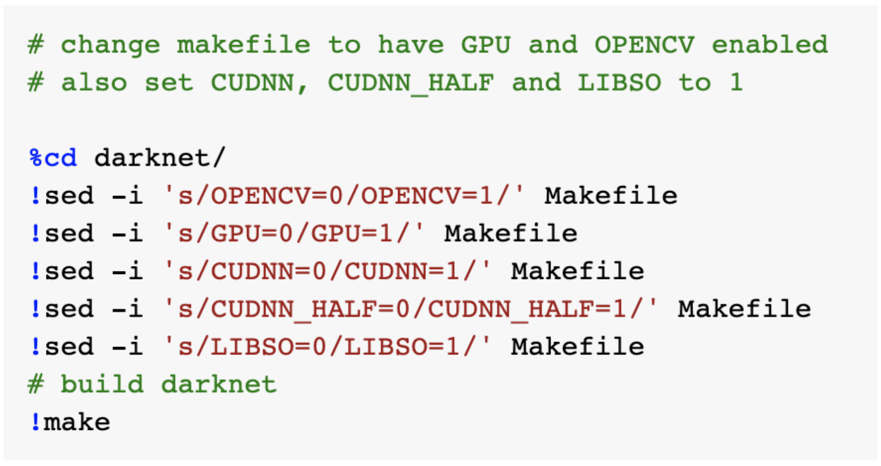
Download the pre-trained weight of yolov4 stored in the directory ~/rungruk_yolov4/darknet where the weight to be used as the pre-trained weight is yolov4.conv.137 The command used to download the pre-trained weight value is !wget ~url as shown in Figure 27.



```
[ ]  # Download the yolov4 pre-trained weights file
     !wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

Figure 27: yolov4-custom.cfg

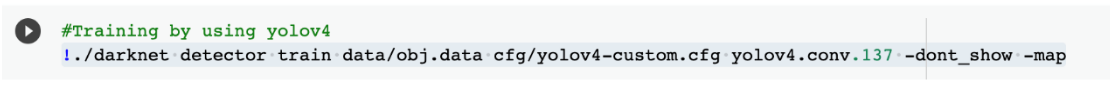
Modify the values in the Makefile to enable OpenCV and enable GPU to use the GPU to train the model using !sed -i 's/OPENCV=0/OPENCV=1/' Makefile.
and !sed -i 's/GPU=0/GPU=1/' Makefile and build darknet using !make command to prepare to train model of YOLOv4 as shown in Figure 28.



```
# change makefile to have GPU and OPENCV enabled
# also set CUDNN, CUDNN_HALF and LIBSO to 1

%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
# build darknet
!make
```

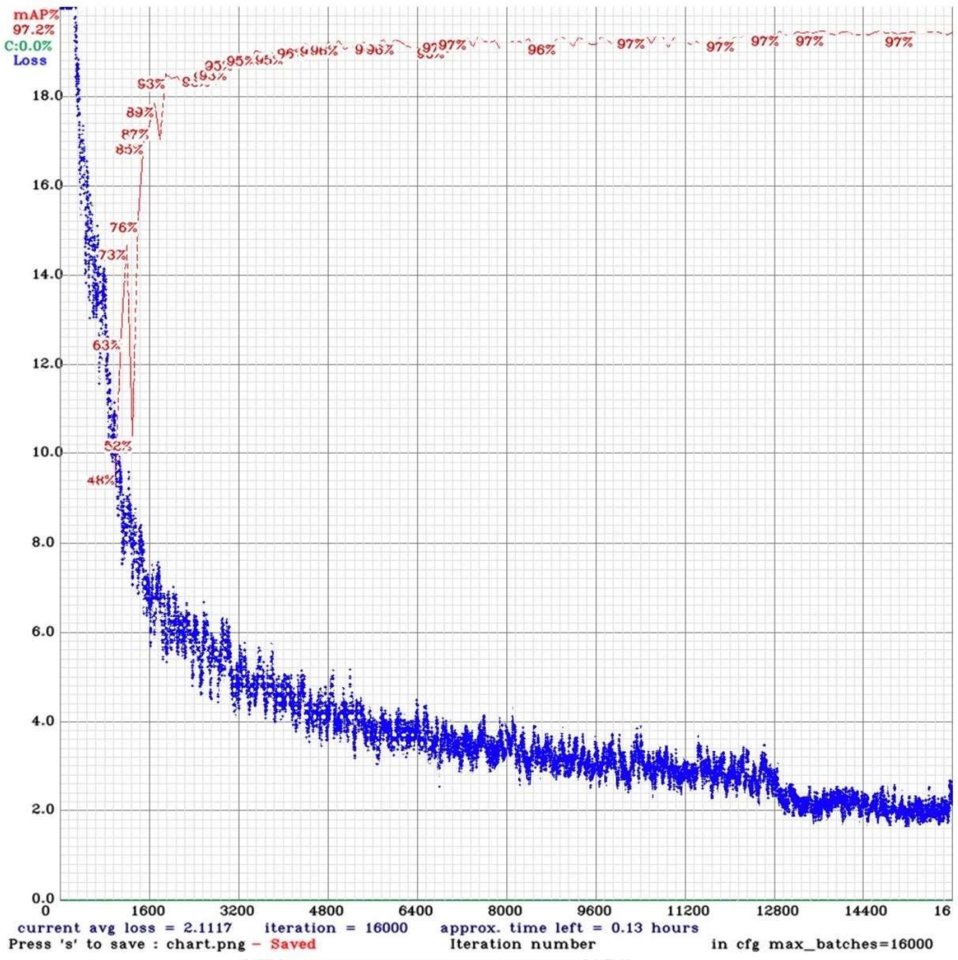
Figure 28: Code for enable OpenCV and GPU

After everything is ready, run the command !./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show –map as shown in Figure 29. To bring dataset and file (* .cfg) that adjusts the parameters and the pre-trained weight together to train the data, where model performance was monitored with the values of loss as shown in Figure 30. The completion of training generates the output file (*.weight). that will be used in the next testing phase



```
#Training by using yolov4
!./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -map
```

Figure 29: Code for trainning model using darknet



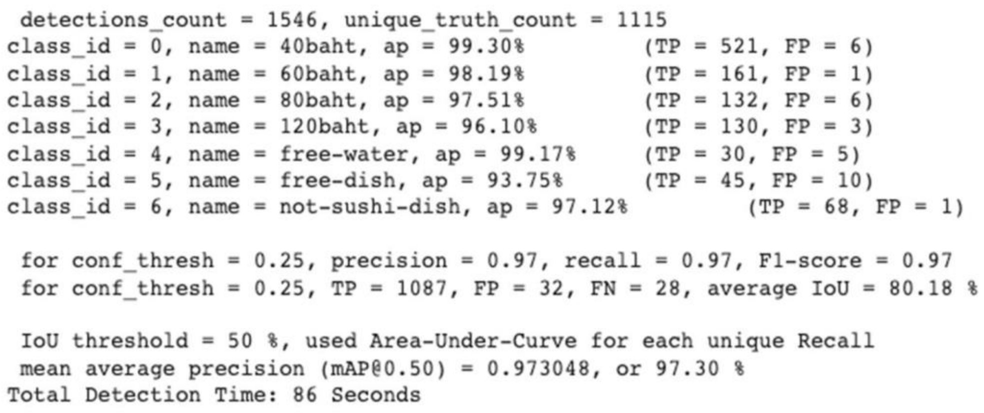
```
current avg loss = 2.1117    iteration = 16000    approx. time left = 0.13 hours
Press 's' to save : chart.png - Saved                            in cfg max_batches=16000
```

Figure 30: Loss graph of 16,000 epochs of training.

4.1.5 Model Performance Evaluation

The values (*.weight) and files (*.cfg) were combined at this stage to detect sushi plates with the trained model. The confidence threshold is set to 0.4 in this work. The successfully detected object class will then be notified by a bounding box associated with the confidence score not lower than 0.4. Figure 31 reports the model performance evaluated with the test image dataset separated from the training set. The performance measures include Precision, Recall, F1-Score, and mAP. And the object detector achieved the values of 97%, 97%, 97%, and 97.3% respectively. The value of precision is calculated by [True positive of all classes / (True positive of all classes, +False positive of all classes)], so we get [1087/(1087+32)] = 97%. Recall is calculated from [True positive of all classes / (True positive of all classes + False Negative of all classes)], so we get [1087/(1087+28)] = 97%. F1-score is

2*[(precision*recall)/(precision+recall)] = 2*[(97*97)/(97+97)] = 97% and The value mAP@0.5 is mean average precision averaged over IOU thresholds in 0.5. It is calculated by taking the average precision of all classes and dividing by the total number of classes (99.30+98.19+97.51+96.10+99.17+93.75+97.12)/7 = 97.30%. The truepositive value is obtained by predicting each class in each figure in the testing set that if the prediction value with IoU is greater than or equal to the threshold value is truepositive.False positive is obtained by predicting each class in each figure in the testing set that if a prediction value has an IoU value less than a threshold value, it is counted as false positive, and False negative is a prediction that cannot predict the class at all in each figure as shown in Figure 32. Figure 33 shows some test images associated with the classification results. The photographs contained 9 characteristics of sushi dishes collected from the selected restaurant including: a) the sushi dishes stacks overlap each other. b) sushi dishes that contain a noise of a sauce plate with a spoon at the top of stack. c) the sushi dishes close-up shot. d) only half-side of sushi dish stack. e) The sushi dishes that contains noises of chopsticks at the top of stack. f) the high stack sushi dishes g) The sushi dishes where the top plate is upside-down and the same color as the beneath. h) The sushi dishes where the top plate is upside-down and the not sushi dish as the beneath. i) The sushi dishes where the top plate is upside-down and different color with the beneath dish. The trained model successfully recognized all 7 object classes. Compared to the results of the CNN model [13], the object detector implemented with YOLOv4 in this work achieves the higher precision, recall, and f1-score values. as shown in Table 2.



```
 detections_count = 1546, unique_truth_count = 1115
class_id = 0, name = 40baht, ap = 99.30%        (TP = 521, FP = 6)
class_id = 1, name = 60baht, ap = 98.19%        (TP = 161, FP = 1)
class_id = 2, name = 80baht, ap = 97.51%        (TP = 132, FP = 6)
class_id = 3, name = 120baht, ap = 96.10%       (TP = 130, FP = 3)
class_id = 4, name = free-water, ap = 99.17%    (TP = 30, FP = 5)
class_id = 5, name = free-dish, ap = 93.75%     (TP = 45, FP = 10)
class_id = 6, name = not-sushi-dish, ap = 97.12%        (TP = 68, FP = 1)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.97, F1-score = 0.97
 for conf_thresh = 0.25, TP = 1087, FP = 32, FN = 28, average IoU = 80.18 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.973048, or 97.30 %
Total Detection Time: 86 Seconds
```

Figure 31: Screen capture showing model performance.

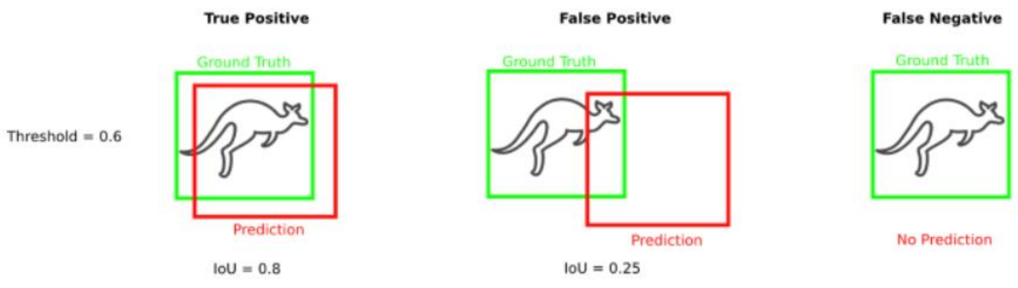


Figure 32: Object detection base on IoU threshold[19].

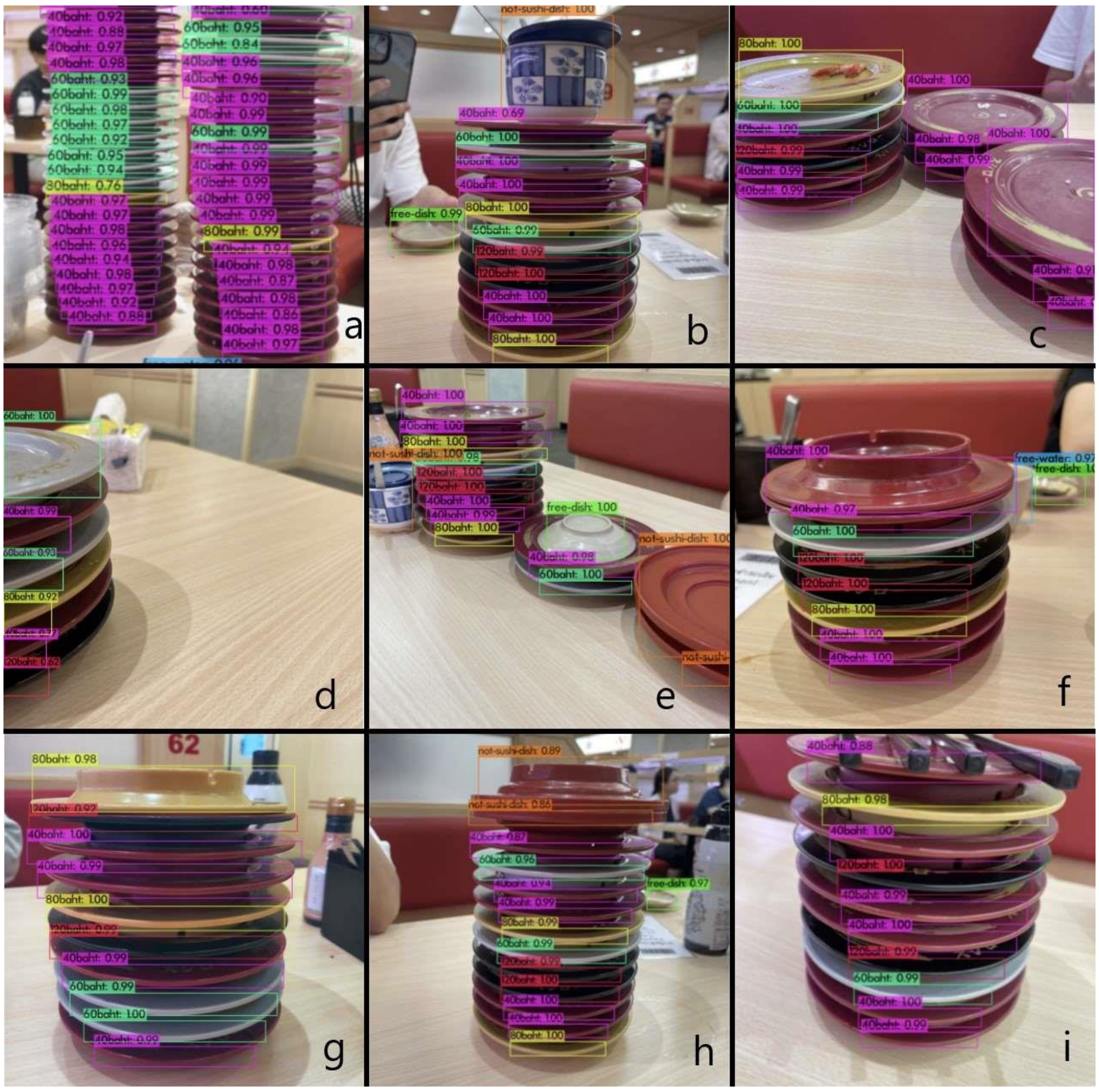


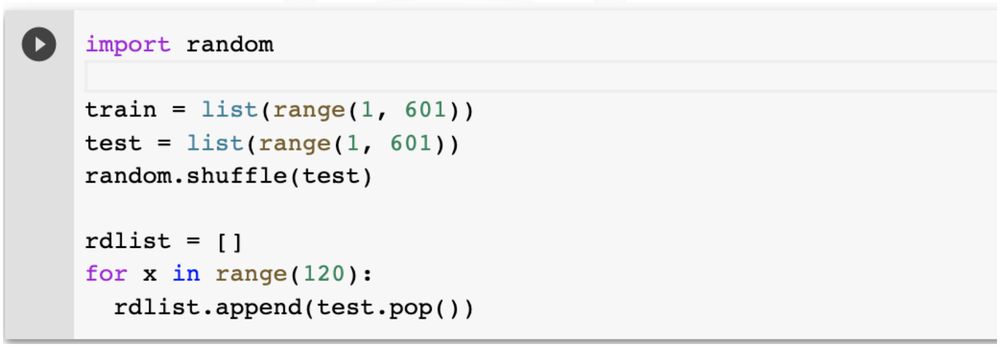
Figure 33: Example of test images and the results of detection.

Table 2: Comparison of model performance of YOLOv4 implemented in this work with CNN model [13]

	Precision	*Recall*	*F1score*	*mAP*	*Accuracy*
CNN	*0.85*	*0.96*	*0.90*	*N/A*	*0.92*
YOLOv4	*0.97*	*0.97*	*0.97*	*97.03*	*N/A*

4.2 Model training by using K Fold Cross validation

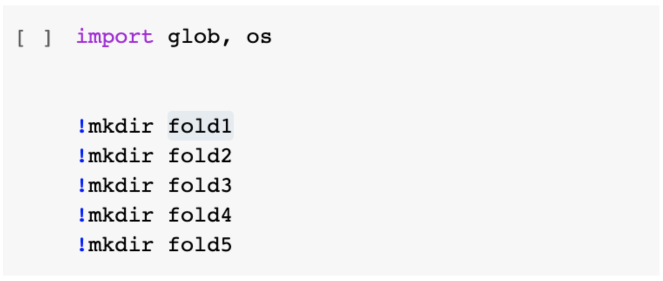
In this step, use k fold cross validation to find which fold have the best fit training set and testing set for the dataset. We chose 5 fold because we will divide the training set 80% and testing set 20%. The dataset has 600 images, so if k = 5 is selected, Every image in the dataset is a testing set at least once, wherein the dataset the image file names are numbers 1-600 in sequence. And the way to split the dataset into testing set. First is using the python library random to randomize the images out of order. After randomization, the images will be stored on the stack. After that, 120 images will be popped from the stack and stored in a new list as shown in Figure 34.



```
import random

train = list(range(1, 601))
test = list(range(1, 601))
random.shuffle(test)

rdlist = []
for x in range(120):
  rdlist.append(test.pop())
```

Figure 34: Python code for split data in k fold

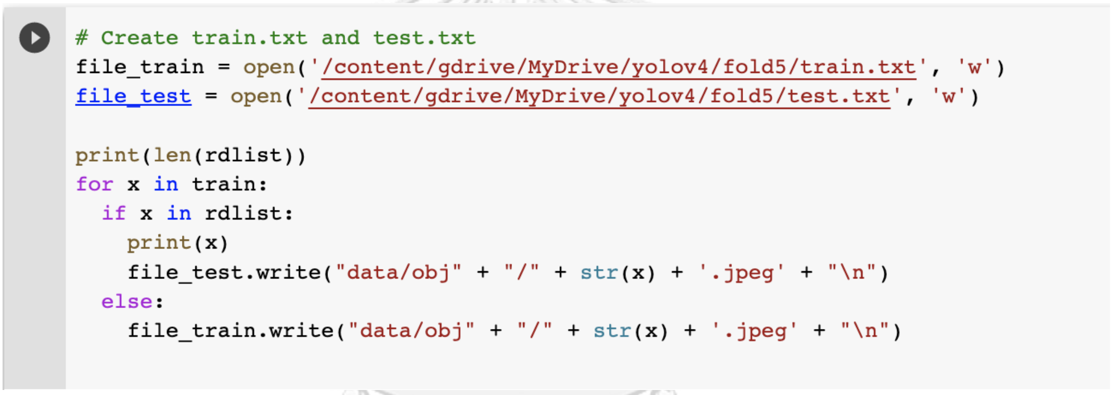
Create a directory to store the training set and testing set of each fold using the command mkdir directory_name as shown in Figure 35 and create a total of 5 directories.



```
[ ]  import glob, os


     !mkdir fold1
     !mkdir fold2
     !mkdir fold3
     !mkdir fold4
     !mkdir fold5
```

Figure 35: Code for create directory command

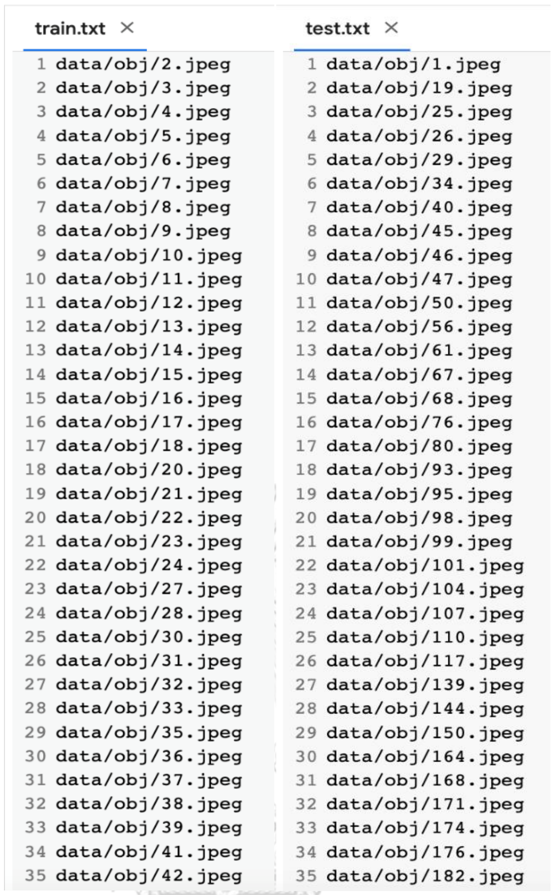
After that, take the list obtained from pop stack 120 to write to file test.txt and put 480 images that are not in test.txt in train.txt as shown in Figure 36. The result is the file train.txt and test.txt that will be in the directory of each fold that will have a unique number of test sets in each fold.



```
# Create train.txt and test.txt
file_train = open('/content/gdrive/MyDrive/yolov4/fold5/train.txt', 'w')
file_test = open('/content/gdrive/MyDrive/yolov4/fold5/test.txt', 'w')

print(len(rdlist))
for x in train:
  if x in rdlist:
    print(x)
    file_test.write("data/obj" + "/" + str(x) + '.jpeg' + "\n")
  else:
    file_train.write("data/obj" + "/" + str(x) + '.jpeg' + "\n")
```

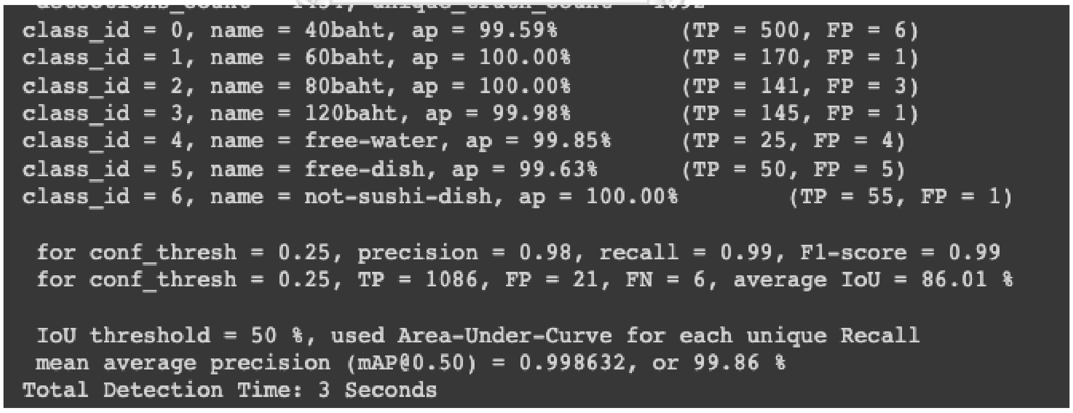
Figure 36: Python code for write train.txt and test.txt in each fold

After that, training dataset of each fold using model yolov4. Starting from Fold 1, training set and testing set as shown in Figure 37. Bring training set and testing set of Fold 1 to training and config (*.cfg) file similar with 4.1.4, the result from training data Fold 1 is a confusion matrix with precision = 98%, recall = 99%, F1-Score = 99% and mAP@0.5 = 99.86% as shown in the figure. at 38



```
train.txt                     test.txt
data/obj/2.jpeg               data/obj/1.jpeg
data/obj/3.jpeg               data/obj/19.jpeg
data/obj/4.jpeg               data/obj/25.jpeg
data/obj/5.jpeg               data/obj/26.jpeg
data/obj/6.jpeg               data/obj/29.jpeg
data/obj/7.jpeg               data/obj/34.jpeg
data/obj/8.jpeg               data/obj/40.jpeg
data/obj/9.jpeg               data/obj/45.jpeg
data/obj/10.jpeg              data/obj/46.jpeg
data/obj/11.jpeg              data/obj/47.jpeg
data/obj/12.jpeg              data/obj/50.jpeg
data/obj/13.jpeg              data/obj/56.jpeg
data/obj/14.jpeg              data/obj/61.jpeg
data/obj/15.jpeg              data/obj/67.jpeg
data/obj/16.jpeg              data/obj/68.jpeg
data/obj/17.jpeg              data/obj/76.jpeg
data/obj/18.jpeg              data/obj/80.jpeg
data/obj/20.jpeg              data/obj/93.jpeg
data/obj/21.jpeg              data/obj/95.jpeg
data/obj/22.jpeg              data/obj/98.jpeg
data/obj/23.jpeg              data/obj/99.jpeg
data/obj/24.jpeg              data/obj/101.jpeg
data/obj/27.jpeg              data/obj/104.jpeg
data/obj/28.jpeg              data/obj/107.jpeg
data/obj/30.jpeg              data/obj/110.jpeg
data/obj/31.jpeg              data/obj/117.jpeg
data/obj/32.jpeg              data/obj/139.jpeg
data/obj/33.jpeg              data/obj/144.jpeg
data/obj/35.jpeg              data/obj/150.jpeg
data/obj/36.jpeg              data/obj/164.jpeg
data/obj/37.jpeg              data/obj/168.jpeg
data/obj/38.jpeg              data/obj/171.jpeg
data/obj/39.jpeg              data/obj/174.jpeg
data/obj/41.jpeg              data/obj/176.jpeg
data/obj/42.jpeg              data/obj/182.jpeg
```

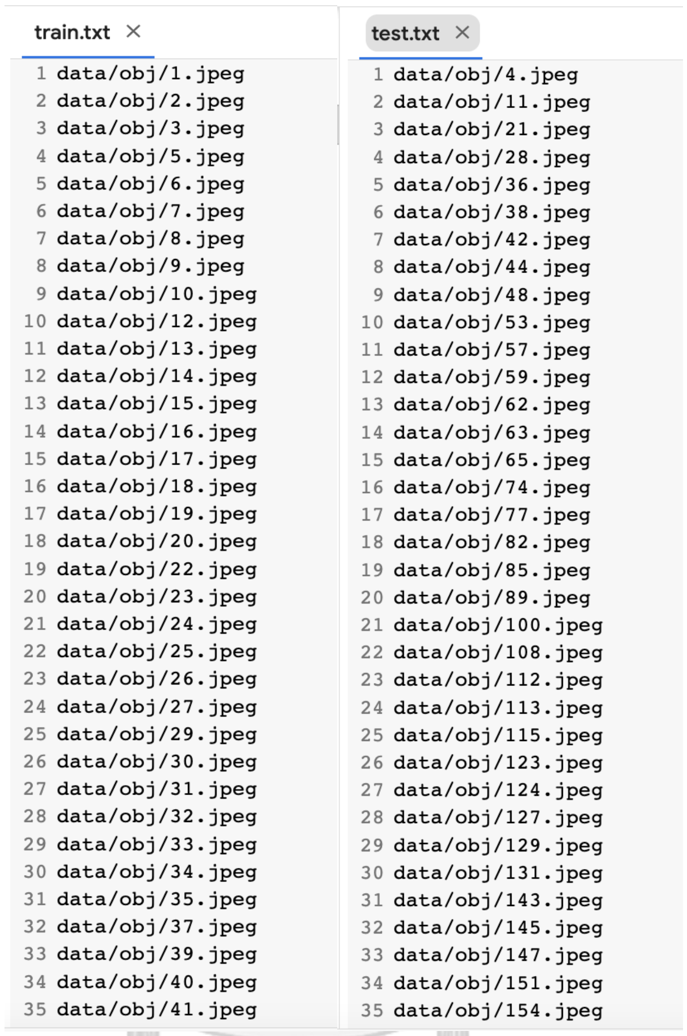
Figure 37: Screen capture Train.txt and Test.txt in Fold1



```
class_id = 0, name = 40baht, ap = 99.59%        (TP = 500, FP = 6)
class_id = 1, name = 60baht, ap = 100.00%       (TP = 170, FP = 1)
class_id = 2, name = 80baht, ap = 100.00%       (TP = 141, FP = 3)
class_id = 3, name = 120baht, ap = 99.98%       (TP = 145, FP = 1)
class_id = 4, name = free-water, ap = 99.85%    (TP = 25, FP = 4)
class_id = 5, name = free-dish, ap = 99.63%     (TP = 50, FP = 5)
class_id = 6, name = not-sushi-dish, ap = 100.00%        (TP = 55, FP = 1)

 for conf_thresh = 0.25, precision = 0.98, recall = 0.99, F1-score = 0.99
 for conf_thresh = 0.25, TP = 1086, FP = 21, FN = 6, average IoU = 86.01 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.998632, or 99.86 %
Total Detection Time: 3 Seconds
```

Figure 38: Screen capture showing model performance in Fold1.

Fold 2 training set and testing set as shown in Figure 39. Bring the training set and testing set of Fold 2 to training and config (*.cfg) file like in 4.1.4. Figure 40 shows the results from train data of Fold 2 is obtained with precision = 96%, recall = 97%, F1-Score = 96% and mAP@0.5 = 95.10%.



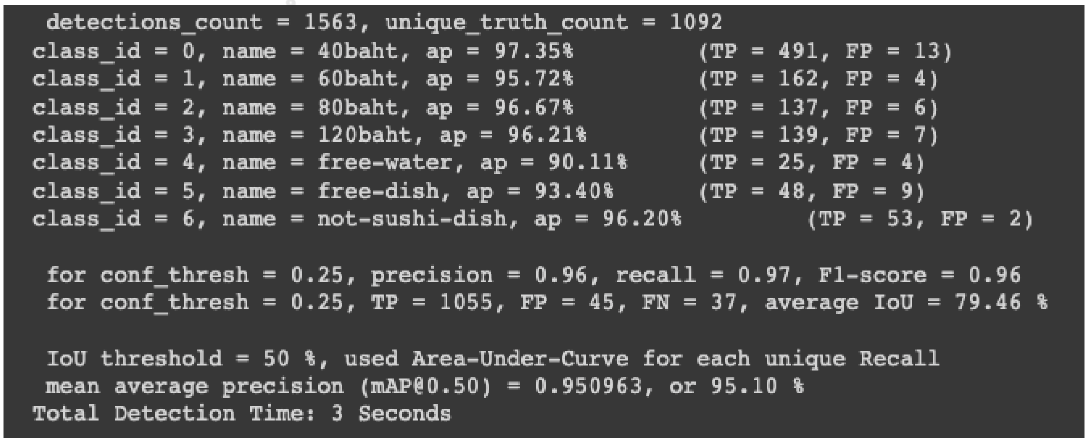
train.txt

```
data/obj/1.jpeg
data/obj/2.jpeg
data/obj/3.jpeg
data/obj/5.jpeg
data/obj/6.jpeg
data/obj/7.jpeg
data/obj/8.jpeg
data/obj/9.jpeg
data/obj/10.jpeg
data/obj/12.jpeg
data/obj/13.jpeg
data/obj/14.jpeg
data/obj/15.jpeg
data/obj/16.jpeg
data/obj/17.jpeg
data/obj/18.jpeg
data/obj/19.jpeg
data/obj/20.jpeg
data/obj/22.jpeg
data/obj/23.jpeg
data/obj/24.jpeg
data/obj/25.jpeg
data/obj/26.jpeg
data/obj/27.jpeg
data/obj/29.jpeg
data/obj/30.jpeg
data/obj/31.jpeg
data/obj/32.jpeg
data/obj/33.jpeg
data/obj/34.jpeg
data/obj/35.jpeg
data/obj/37.jpeg
data/obj/39.jpeg
data/obj/40.jpeg
data/obj/41.jpeg
```

test.txt

```
data/obj/4.jpeg
data/obj/11.jpeg
data/obj/21.jpeg
data/obj/28.jpeg
data/obj/36.jpeg
data/obj/38.jpeg
data/obj/42.jpeg
data/obj/44.jpeg
data/obj/48.jpeg
data/obj/53.jpeg
data/obj/57.jpeg
data/obj/59.jpeg
data/obj/62.jpeg
data/obj/63.jpeg
data/obj/65.jpeg
data/obj/74.jpeg
data/obj/77.jpeg
data/obj/82.jpeg
data/obj/85.jpeg
data/obj/89.jpeg
data/obj/100.jpeg
data/obj/108.jpeg
data/obj/112.jpeg
data/obj/113.jpeg
data/obj/115.jpeg
data/obj/123.jpeg
data/obj/124.jpeg
data/obj/127.jpeg
data/obj/129.jpeg
data/obj/131.jpeg
data/obj/143.jpeg
data/obj/145.jpeg
data/obj/147.jpeg
data/obj/151.jpeg
data/obj/154.jpeg
```

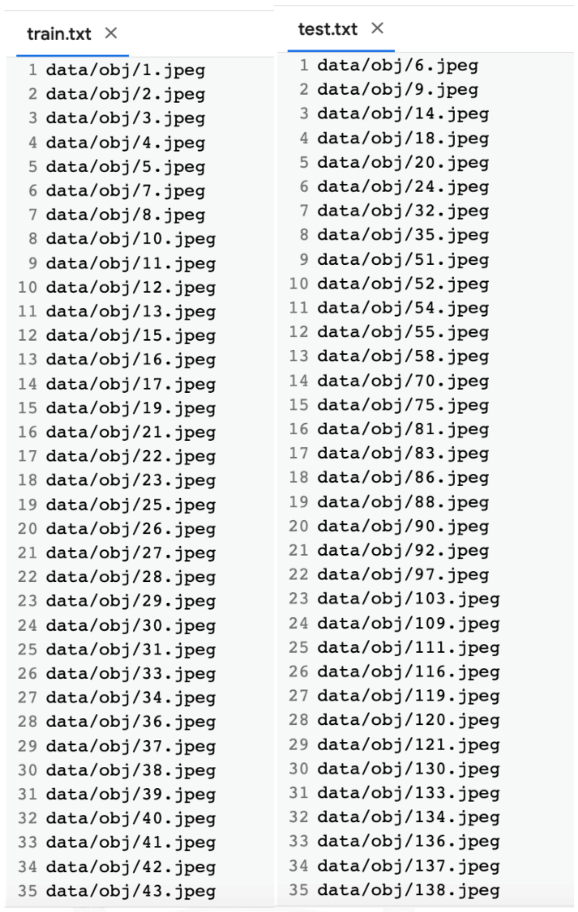
Figure 39: Screen capture Train.txt and Test.txt in Fold2



```
 detections_count = 1563, unique_truth_count = 1092
class_id = 0, name = 40baht, ap = 97.35%         (TP = 491, FP = 13)
class_id = 1, name = 60baht, ap = 95.72%         (TP = 162, FP = 4)
class_id = 2, name = 80baht, ap = 96.67%         (TP = 137, FP = 6)
class_id = 3, name = 120baht, ap = 96.21%        (TP = 139, FP = 7)
class_id = 4, name = free-water, ap = 90.11%     (TP = 25, FP = 4)
class_id = 5, name = free-dish, ap = 93.40%      (TP = 48, FP = 9)
class_id = 6, name = not-sushi-dish, ap = 96.20%         (TP = 53, FP = 2)

 for conf_thresh = 0.25, precision = 0.96, recall = 0.97, F1-score = 0.96
 for conf_thresh = 0.25, TP = 1055, FP = 45, FN = 37, average IoU = 79.46 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.950963, or 95.10 %
Total Detection Time: 3 Seconds
```

Figure 40: Screen capture showing model performance in Fold2.

Fold 3 training set and testing set as shown in Figure 41. Bring the training set and testing set of Fold 3 to training and config (*.cfg) file like in 4.1.4. Figure 42 shows the results obtained from train data of Fold 3 is obtained with precision = 91%, recall = 95%, F1-Score = 93% and mAP@0.5 = 94.34%.



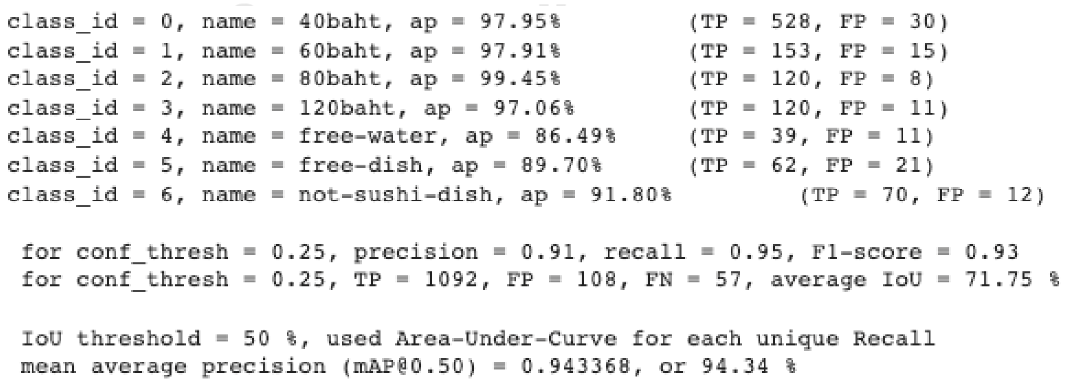
train.txt ×

```
data/obj/1.jpeg
data/obj/2.jpeg
data/obj/3.jpeg
data/obj/4.jpeg
data/obj/5.jpeg
data/obj/7.jpeg
data/obj/8.jpeg
data/obj/10.jpeg
data/obj/11.jpeg
data/obj/12.jpeg
data/obj/13.jpeg
data/obj/15.jpeg
data/obj/16.jpeg
data/obj/17.jpeg
data/obj/19.jpeg
data/obj/21.jpeg
data/obj/22.jpeg
data/obj/23.jpeg
data/obj/25.jpeg
data/obj/26.jpeg
data/obj/27.jpeg
data/obj/28.jpeg
data/obj/29.jpeg
data/obj/30.jpeg
data/obj/31.jpeg
data/obj/33.jpeg
data/obj/34.jpeg
data/obj/36.jpeg
data/obj/37.jpeg
data/obj/38.jpeg
data/obj/39.jpeg
data/obj/40.jpeg
data/obj/41.jpeg
data/obj/42.jpeg
data/obj/43.jpeg
```

test.txt ×

```
data/obj/6.jpeg
data/obj/9.jpeg
data/obj/14.jpeg
data/obj/18.jpeg
data/obj/20.jpeg
data/obj/24.jpeg
data/obj/32.jpeg
data/obj/35.jpeg
data/obj/51.jpeg
data/obj/52.jpeg
data/obj/54.jpeg
data/obj/55.jpeg
data/obj/58.jpeg
data/obj/70.jpeg
data/obj/75.jpeg
data/obj/81.jpeg
data/obj/83.jpeg
data/obj/86.jpeg
data/obj/88.jpeg
data/obj/90.jpeg
data/obj/92.jpeg
data/obj/97.jpeg
data/obj/103.jpeg
data/obj/109.jpeg
data/obj/111.jpeg
data/obj/116.jpeg
data/obj/119.jpeg
data/obj/120.jpeg
data/obj/121.jpeg
data/obj/130.jpeg
data/obj/133.jpeg
data/obj/134.jpeg
data/obj/136.jpeg
data/obj/137.jpeg
data/obj/138.jpeg
```

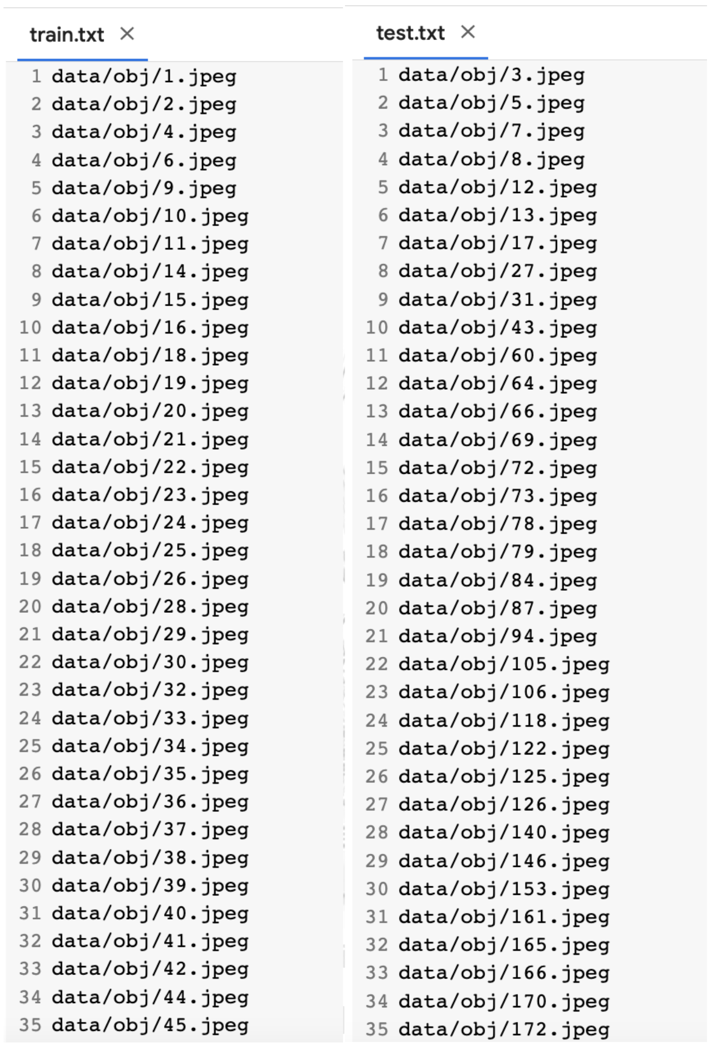
Figure 41: Screen capture Train.txt and Test.txt in Fold3



```
class_id = 0, name = 40baht, ap = 97.95%          (TP = 528, FP = 30)
class_id = 1, name = 60baht, ap = 97.91%          (TP = 153, FP = 15)
class_id = 2, name = 80baht, ap = 99.45%          (TP = 120, FP = 8)
class_id = 3, name = 120baht, ap = 97.06%         (TP = 120, FP = 11)
class_id = 4, name = free-water, ap = 86.49%      (TP = 39, FP = 11)
class_id = 5, name = free-dish, ap = 89.70%       (TP = 62, FP = 21)
class_id = 6, name = not-sushi-dish, ap = 91.80%          (TP = 70, FP = 12)

 for conf_thresh = 0.25, precision = 0.91, recall = 0.95, F1-score = 0.93
 for conf_thresh = 0.25, TP = 1092, FP = 108, FN = 57, average IoU = 71.75 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.943368, or 94.34 %
```

Figure 42: Screen capture showing model performance in Fold3.

Fold 4 training set and testing set as shown in Figure 43. Bring the training set and testing set of Fold 4 to training and config (*.cfg) file like in 4.1.4. Figure 44 shows the results from train data of Fold 4 is obtained with precision = 97%, recall = 99%, F1-Score = 98% and mAP@0.5 = 99.67%.



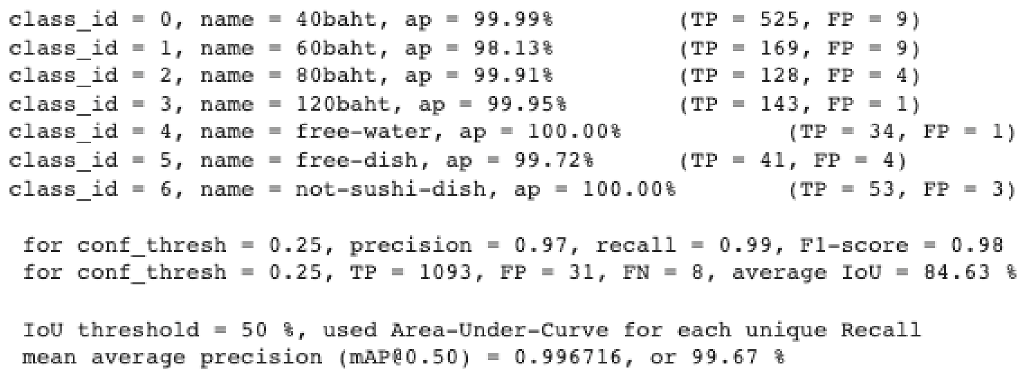
train.txt

```
data/obj/1.jpeg
data/obj/2.jpeg
data/obj/4.jpeg
data/obj/6.jpeg
data/obj/9.jpeg
data/obj/10.jpeg
data/obj/11.jpeg
data/obj/14.jpeg
data/obj/15.jpeg
data/obj/16.jpeg
data/obj/18.jpeg
data/obj/19.jpeg
data/obj/20.jpeg
data/obj/21.jpeg
data/obj/22.jpeg
data/obj/23.jpeg
data/obj/24.jpeg
data/obj/25.jpeg
data/obj/26.jpeg
data/obj/28.jpeg
data/obj/29.jpeg
data/obj/30.jpeg
data/obj/32.jpeg
data/obj/33.jpeg
data/obj/34.jpeg
data/obj/35.jpeg
data/obj/36.jpeg
data/obj/37.jpeg
data/obj/38.jpeg
data/obj/39.jpeg
data/obj/40.jpeg
data/obj/41.jpeg
data/obj/42.jpeg
data/obj/44.jpeg
data/obj/45.jpeg
```

test.txt

```
data/obj/3.jpeg
data/obj/5.jpeg
data/obj/7.jpeg
data/obj/8.jpeg
data/obj/12.jpeg
data/obj/13.jpeg
data/obj/17.jpeg
data/obj/27.jpeg
data/obj/31.jpeg
data/obj/43.jpeg
data/obj/60.jpeg
data/obj/64.jpeg
data/obj/66.jpeg
data/obj/69.jpeg
data/obj/72.jpeg
data/obj/73.jpeg
data/obj/78.jpeg
data/obj/79.jpeg
data/obj/84.jpeg
data/obj/87.jpeg
data/obj/94.jpeg
data/obj/105.jpeg
data/obj/106.jpeg
data/obj/118.jpeg
data/obj/122.jpeg
data/obj/125.jpeg
data/obj/126.jpeg
data/obj/140.jpeg
data/obj/146.jpeg
data/obj/153.jpeg
data/obj/161.jpeg
data/obj/165.jpeg
data/obj/166.jpeg
data/obj/170.jpeg
data/obj/172.jpeg
```

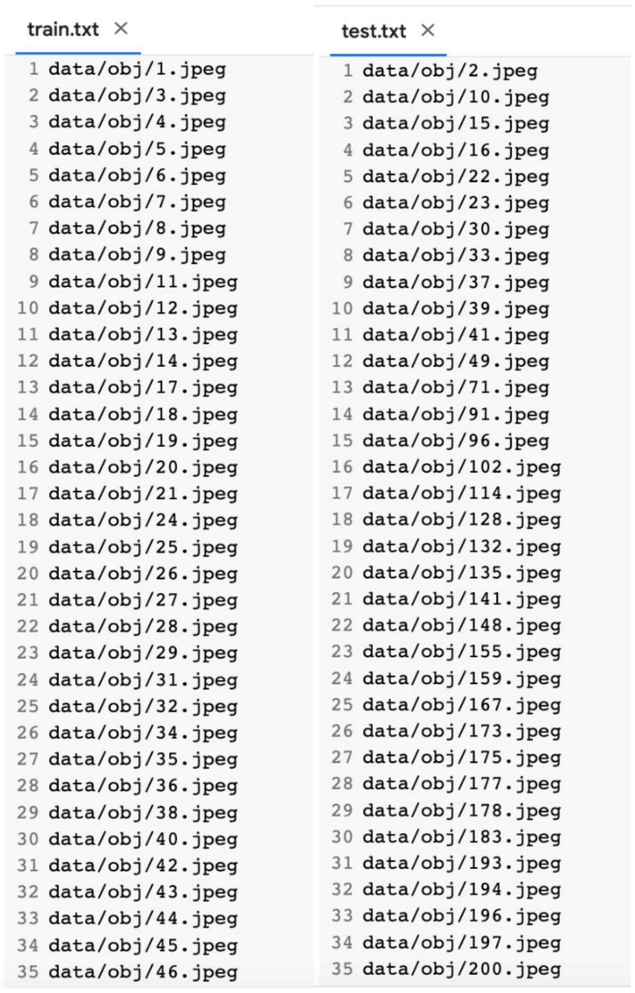
Figure 43: Screen capture Train.txt and Test.txt in Fold4



```
class_id = 0, name = 40baht, ap = 99.99%         (TP = 525, FP = 9)
class_id = 1, name = 60baht, ap = 98.13%         (TP = 169, FP = 9)
class_id = 2, name = 80baht, ap = 99.91%         (TP = 128, FP = 4)
class_id = 3, name = 120baht, ap = 99.95%        (TP = 143, FP = 1)
class_id = 4, name = free-water, ap = 100.00%            (TP = 34, FP = 1)
class_id = 5, name = free-dish, ap = 99.72%      (TP = 41, FP = 4)
class_id = 6, name = not-sushi-dish, ap = 100.00%        (TP = 53, FP = 3)

 for conf_thresh = 0.25, precision = 0.97, recall = 0.99, F1-score = 0.98
 for conf_thresh = 0.25, TP = 1093, FP = 31, FN = 8, average IoU = 84.63 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.996716, or 99.67 %
```

Figure 44: Screen capture showing model performance in Fold4.

And the last fold is Fold5. Fold 5 training set and testing set as shown in Figure 45. Take the training set and testing set of Fold 5 for traning and config file (*.cfg) as in 4.1.4. Figure 46 shows the results from training data. Fold 5 is obtained with precision = 95%, recall = 97%, F1-Score = 96% and mAP@0.5 = 96.47%.

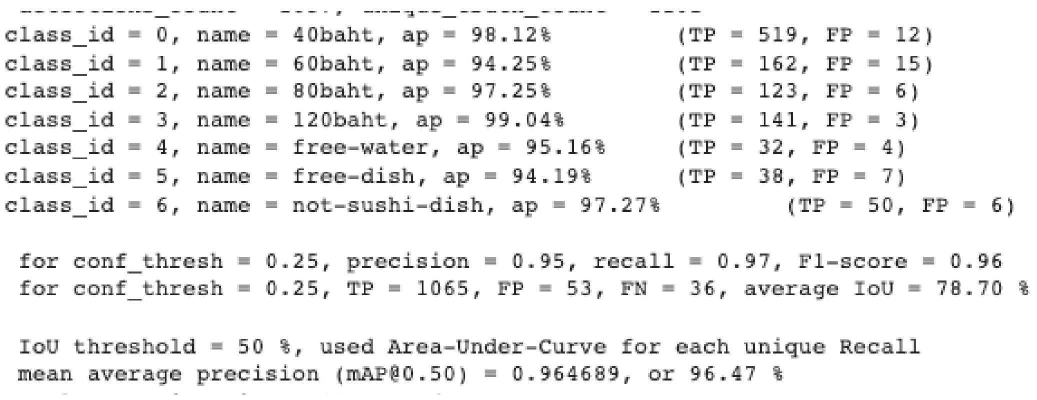

train.txt

```
data/obj/1.jpeg
data/obj/3.jpeg
data/obj/4.jpeg
data/obj/5.jpeg
data/obj/6.jpeg
data/obj/7.jpeg
data/obj/8.jpeg
data/obj/9.jpeg
data/obj/11.jpeg
data/obj/12.jpeg
data/obj/13.jpeg
data/obj/14.jpeg
data/obj/17.jpeg
data/obj/18.jpeg
data/obj/19.jpeg
data/obj/20.jpeg
data/obj/21.jpeg
data/obj/24.jpeg
data/obj/25.jpeg
data/obj/26.jpeg
data/obj/27.jpeg
data/obj/28.jpeg
data/obj/29.jpeg
data/obj/31.jpeg
data/obj/32.jpeg
data/obj/34.jpeg
data/obj/35.jpeg
data/obj/36.jpeg
data/obj/38.jpeg
data/obj/40.jpeg
data/obj/42.jpeg
data/obj/43.jpeg
data/obj/44.jpeg
data/obj/45.jpeg
data/obj/46.jpeg
```

test.txt

```
data/obj/2.jpeg
data/obj/10.jpeg
data/obj/15.jpeg
data/obj/16.jpeg
data/obj/22.jpeg
data/obj/23.jpeg
data/obj/30.jpeg
data/obj/33.jpeg
data/obj/37.jpeg
data/obj/39.jpeg
data/obj/41.jpeg
data/obj/49.jpeg
data/obj/71.jpeg
data/obj/91.jpeg
data/obj/96.jpeg
data/obj/102.jpeg
data/obj/114.jpeg
data/obj/128.jpeg
data/obj/132.jpeg
data/obj/135.jpeg
data/obj/141.jpeg
data/obj/148.jpeg
data/obj/155.jpeg
data/obj/159.jpeg
data/obj/167.jpeg
data/obj/173.jpeg
data/obj/175.jpeg
data/obj/177.jpeg
data/obj/178.jpeg
data/obj/183.jpeg
data/obj/193.jpeg
data/obj/194.jpeg
data/obj/196.jpeg
data/obj/197.jpeg
data/obj/200.jpeg
```

Figure 45: Screen capture Train.txt and Test.txt in Fold5.



```
class_id = 0, name = 40baht, ap = 98.12%          (TP = 519, FP = 12)
class_id = 1, name = 60baht, ap = 94.25%          (TP = 162, FP = 15)
class_id = 2, name = 80baht, ap = 97.25%          (TP = 123, FP = 6)
class_id = 3, name = 120baht, ap = 99.04%         (TP = 141, FP = 3)
class_id = 4, name = free-water, ap = 95.16%      (TP = 32, FP = 4)
class_id = 5, name = free-dish, ap = 94.19%       (TP = 38, FP = 7)
class_id = 6, name = not-sushi-dish, ap = 97.27%         (TP = 50, FP = 6)

 for conf_thresh = 0.25, precision = 0.95, recall = 0.97, F1-score = 0.96
 for conf_thresh = 0.25, TP = 1065, FP = 53, FN = 36, average IoU = 78.70 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.964689, or 96.47 %
```

Figure 46: Screen capture showing model performance in Fold5.

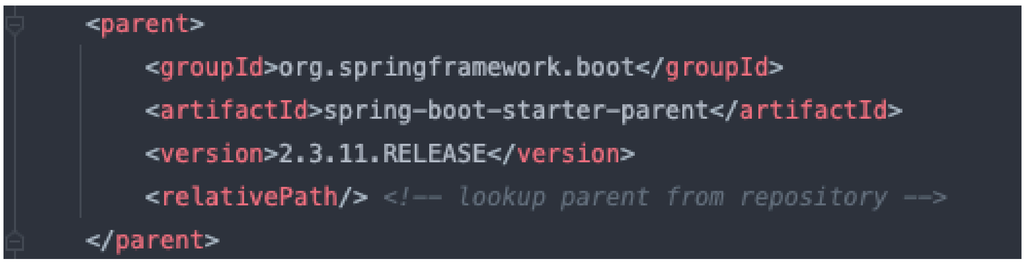
Table 3 shows the results from k fold cross validation. Fold1 is the fold with the highest precision, recall, f1-score and mAP values compared to other folds. The average results were precision = 95%, recall = 97%, f1-score = 96% and mAP = 97.07. It was observed that the results were similar to the train model in 4.1.4 using a randomized training set and testing set without doing k fold cross validation.

Table 3: Summary and average of performance metrics of 5-fold.

	Precision	*Recall*	*F1-Score*	*mAP*
Fold1	**0.98**	**0.99**	**0.99**	**99.86**
Fold2	0.96	0.97	0.96	95.10
Fold3	0.91	0.95	0.93	94.34
Fold4	0.97	0.99	0.98	99.67
Fold5	0.95	0.97	0.96	96.47
Average	0.95	0.97	0.96	97.09

4.3 API Gateway

The API Gateway uses Spring Boot REST API version 2.3.11. RELEASE to create a controller for API. The Spring Boot library is called by adding the Spring Boot dependency format xml into the pom.xml file as shown in Figure 47.



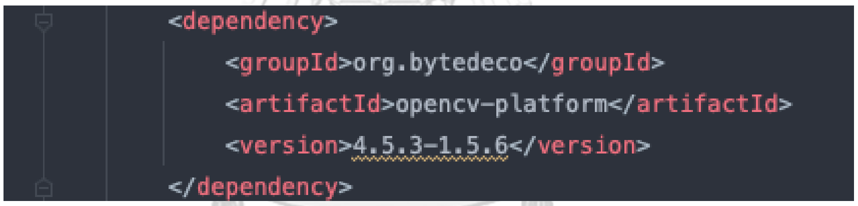
```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.11.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Figure 47: Spring Boot library in pom.xml

4.4 Microservice

4.4.1 Payment service

Payment service uses OpenCV version 4.5.3-1.5.6 . which is maven library as shown in Figure 48. The process of calculating payment is shown in Figure 49. It begins by converting the image to base64 and then decode it to byte-array. It translates the byte-array value into Mat format, which is an OpenCV image variable. Next, it takes the Mat value along with importing the YOLOv4 exercise's values (*.weight) and files (*.cfg) into the readNetFromDarknet function. To detect and classify sushi plates, set the parameters to confThreshold at 0.4 and nmsThreshold at 0.4. The money is then calculated by counting each verifiable type of sushi dish. As illustrated in Figure 50, checking the accuracy of the calculation and the number of sushi plates of each kind using postman to test the API. Request body as image converted to base64 and Response reveals the accuracy of the calculation and the correct number of sushi plates of each type.



```
<dependency>
    <groupId>org.bytedeco</groupId>
    <artifactId>opencv-platform</artifactId>
    <version>4.5.3-1.5.6</version>
</dependency>
```

Figure 48: OpenCV version 4.5.3-1.5.6.

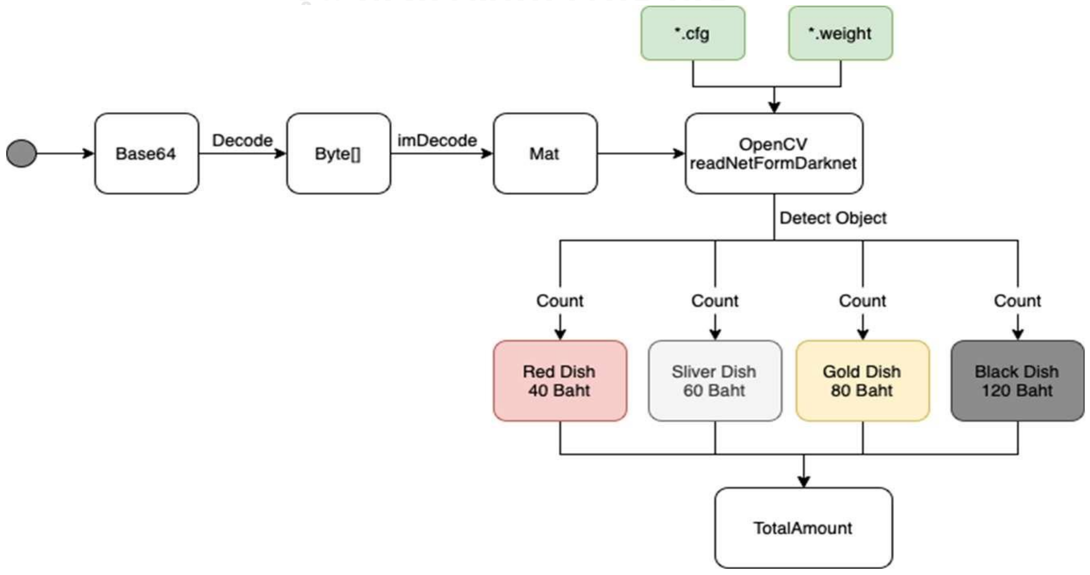
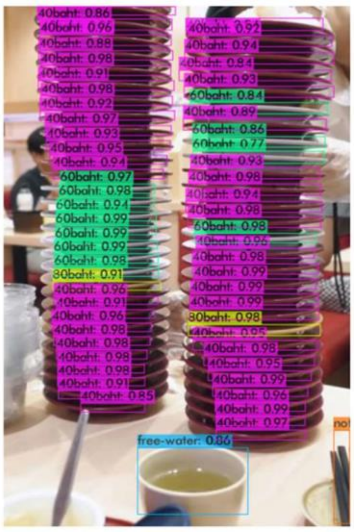
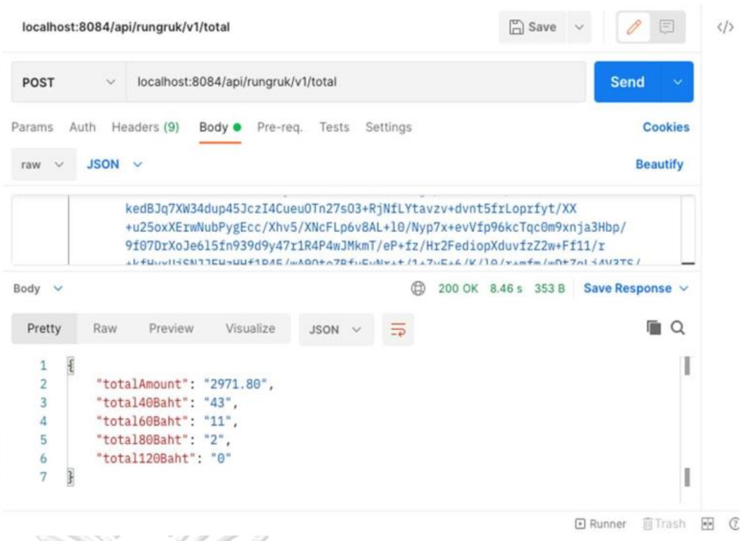


Figure 49: Process flow of payment service.




localhost:8084/api/rungruk/v1/total

POST localhost:8084/api/rungruk/v1/total

Params Auth Headers (9) Body Pre-req. Tests Settings

raw JSON

```
kedBJq7XW34dup45JczI4CueuOTn27sO3+RjNfLYtavzv+dvnt5frLoprfyt/XX
+u25oxXErwNubPygEcc/Xhv5/XNcFLp6v8AL+l0/Nyp7x+evVfp96kcTqc0m9xnja3Hbp/
9f07DrXoJe615fn939d9y47r1R4P4wJMkmT/eP+fz/Hr2FediopXduvfzZ2w+Ff11/r
```

Body 200 OK 8.46 s 353 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "totalAmount": "2971.80",
    "total40Baht": "43",
    "total60Baht": "11",
    "total80Baht": "2",
    "total120Baht": "0"
}
```

Figure 50: Screen capture of postman testing payment service.

4.4.2 Userprofile

The main function of Userprofile service is to store and retrieve customer information that has been subscribed to the database. It is categorized into 3 parts:

1. User_Register is an API that allows customers to apply for membership with the restaurant, where customer information will be stored in the store's database. SignIn function receive a request body from frontEnd 9 values: memberID, email, password, firstname, lastname, phoneNumber, age, birthday. The password stored in the database must be hashed by algorithm PBKDF2 for security purposes. Figure 51 shows the operation of hashing with algorithm PBKDF2, which starts from receiving requestBody password sent as String, which is a String that has not been hashed, and converts String to ByteArray and a salt is set to be used for the hash encryption. After the encryption is complete, the value will be converted to HexString and stored in the database.

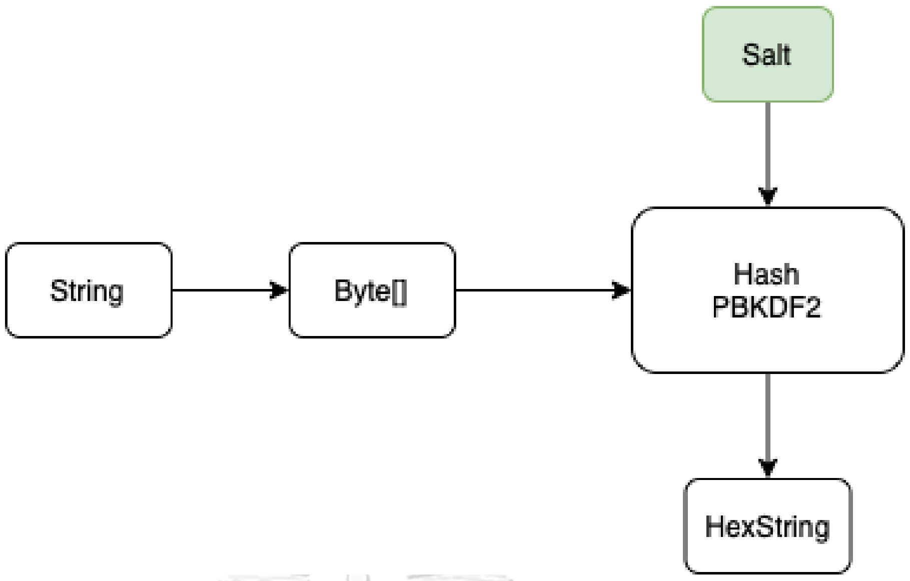


Figure 51: Password encoding step

2. User_SignOut is an API made to provide Customer logout. SignOut function will send a value to update the database in the table so that the user's status is N.

3. User_SignIn is an API made to provide Customer login. This function will get requestBody from frontEnd as 2 values: username , password . the function will take username and password that have been hashed with PBKDF2 to check at database and like both values match in database, will return response back to frontEnd total 7 values is memberID ,firstname, lastname, phoneNumber, age, email, birthday and change status to Y

4.5 Database

The database was created using Structured Query Language (SQL). TRANSACTION_LOG and MEMBER are the only two variables in the database. TRANSACTION_LOG records the details of consumers who visit the conveyor belt sushi restaurant on a daily basis. MEMBER stores information on customers who have signed up to be restaurant members. Table 4 shows the database schema and the datatypes that stored in the table TRANSACTION_LOG. And Table 5 shows the schema of the data and the date type that stored in the MEMBER table.

Table 4: TRANSACTION_LOG table description.

Database name: demo		Table name: TRANSACTION_LOG			
Table description: table that keep customer daily log					
No.	FieldName	Description	DataType	Key	Condition
1	ID	Unique number	VARCHAR(32)	PK	NOT NULL
2	MEMBER_ID	Customer id	VARCHAR(32)	FK	NOT NULL
3	REQUEST_UID	API Request id	VARCHAR(22)		
5	DETAILS	Customer Details	JSON		
6	AMOUNT	Customer amount per bill	VARCHAR(100)		
7	AMOUNT_NET	Amount + (17% of VAT/Service charge)	VARCHAR(100)		
8	BRANCH_NO	Merchant branch number	VARCHAR(4)		
9	TRANSACTION_DATE	Date of log	DATE		
10	CREATE_DATE	Record created date time	TIMESTAMP		

Table 5: MEMBER table description.

Database name: demo	Table name: Member				
Table description: table that keep customer data					
No.	FieldName	Description	DataType	Key	Condition
1	EMAIL	Customer email	VARCHAR(200)	PK	NOT NULL
2	MEMBER_ID	Customer id	VARCHAR(32)	FK	NOT NULL
3	FIRSTNAME	Customer first name	VARCHAR(300)		
5	LASTNAME	Customer last name	VARCHAR(300)		
6	PHONE_NUMBER	Customer phone number	VARCHAR(20)		
7	AGE	Customer age	VARCHAR(4)		
8	PASSWORD	Customer password	VARCHAR(100)		
9	DATE_OF_BIRTH	Customer birthday	DATE		
10	STATUS	Login Status	VARCHAR(1)		
11	CREATE_DATE	Record created date time	TIMESTAMP		

Chapter 5
Conclusion

This thesis presents the application of YOLOv4 object detection to detect and classify sushi dishes, resulting in a new innovation of smart billing for conveyor belt sushi business that will replace the current billing system that uses RFID to calculate the billing system. RFID is costly due to the investment and maintenance of attaching tags to each plate. The exhausted radio-frequency batteries may cause counting error, in addition to increase the electronic waste hazardous to society. The approach also requires manual count to double-check the total number of consumed dishes. The dataset used is a reliable dataset from the Japanese's most famous conveyor belt sushi franchise restaurant in Bangkok, Thailand. and collecting 600 images of sushi plates by myself. Therefore, the detection and classification of sushi plates requires high accuracy and speed. It is suitable for using YOLOv4 in this work because YOLOv4 is a single state objection, which makes it faster than other models and achieves the accuracy similar as the multiple state objection model.

For application in real life, the researcher has created an API that can be applied to any mobile device to calculate payment. The payment API functionality brings a file (*.weight) that is the resulting file from train the YOLOv4 model and integrates it with Spring Boot RESTAPI to create an API that can calculate payment for classification sushi dish

All images in the dataset must be labeled images before being used in the train model. The labeling is a manual 600 images, then split the data into 80% train sets and 20% test sets, and then train the dataset through the YOLOv4 model. The YOLOv4 achieved the values of Precision = 97%, Recall = 97%, F1-Score = 97% and mAP = 97.3% by randomly split dataset in ratio 80:20 and Experiment with K Fold Cross validation using k = 5, found that the mean of confusion metric value equal to Precision = 95%, Recall = 97% , F1-Score = 96% , mAP = 97.07%.

Since the API is still running on localhost, the system management cannot try to handle the massive incoming traffic. Further direction would be applying Cloud

Native and Load Balancer to handle the massive traffic that will help the API support the traffic from the usage better, as well as improve the model performance. And increase the model's ability to be compatible with other types of restaurant dishes other than plates from conveyor belt sushi restaurants.

Appendix

Model Testing Result

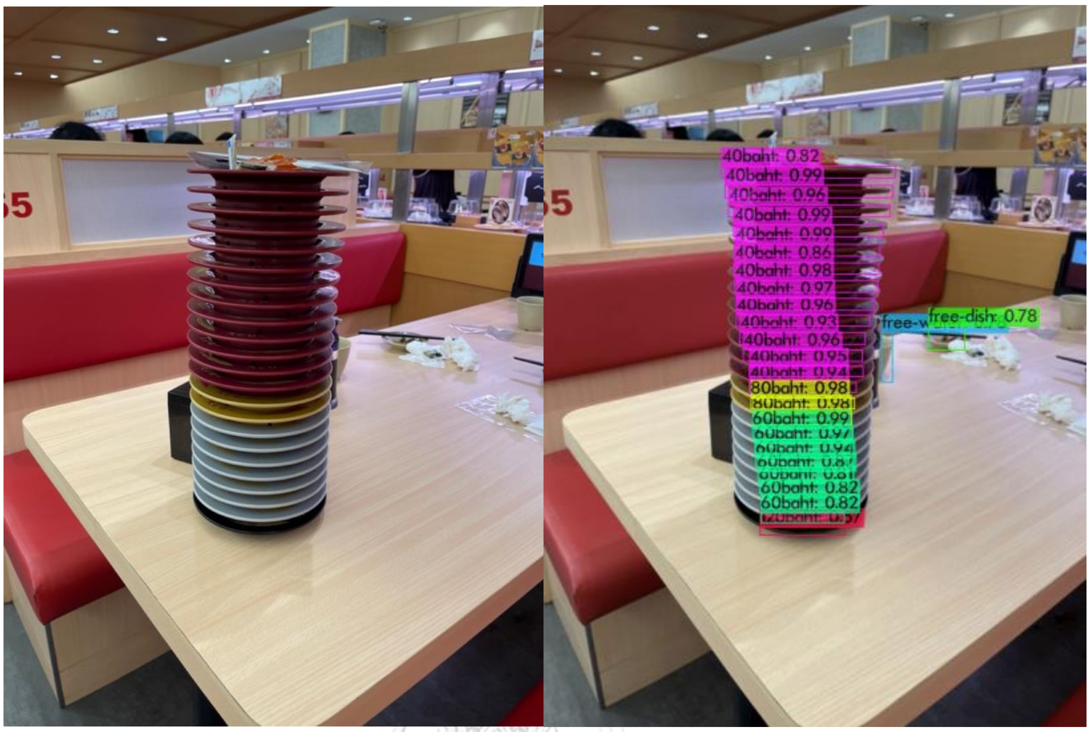


Figure 52: (a) The image in the testset. (b) The result of detection

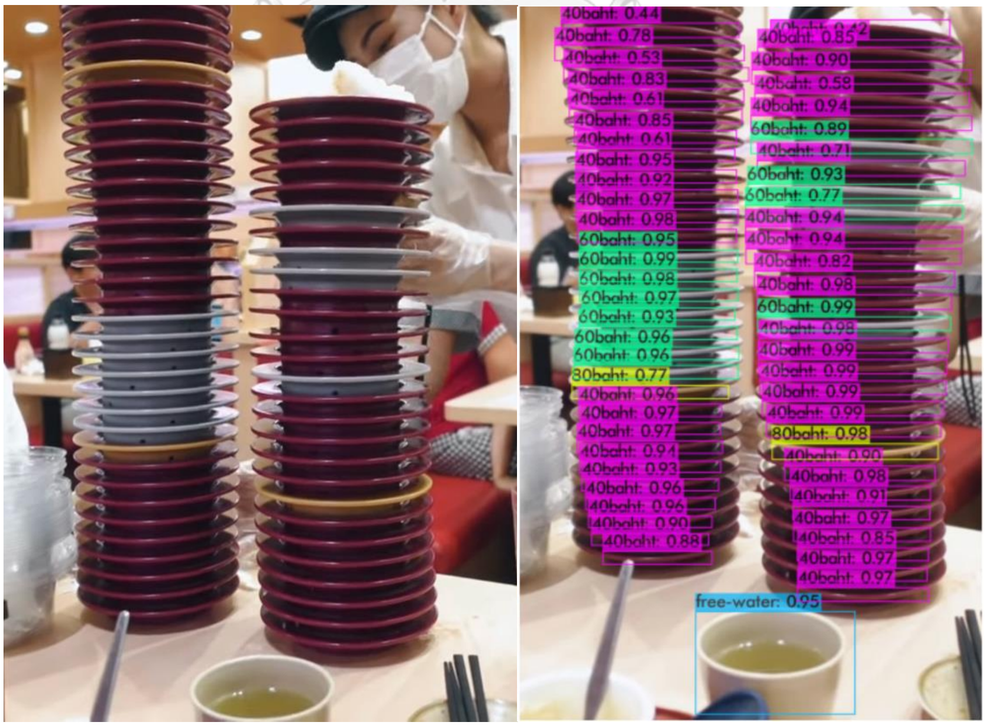


Figure 53: (a) The image in the testset. (b) The result of detection

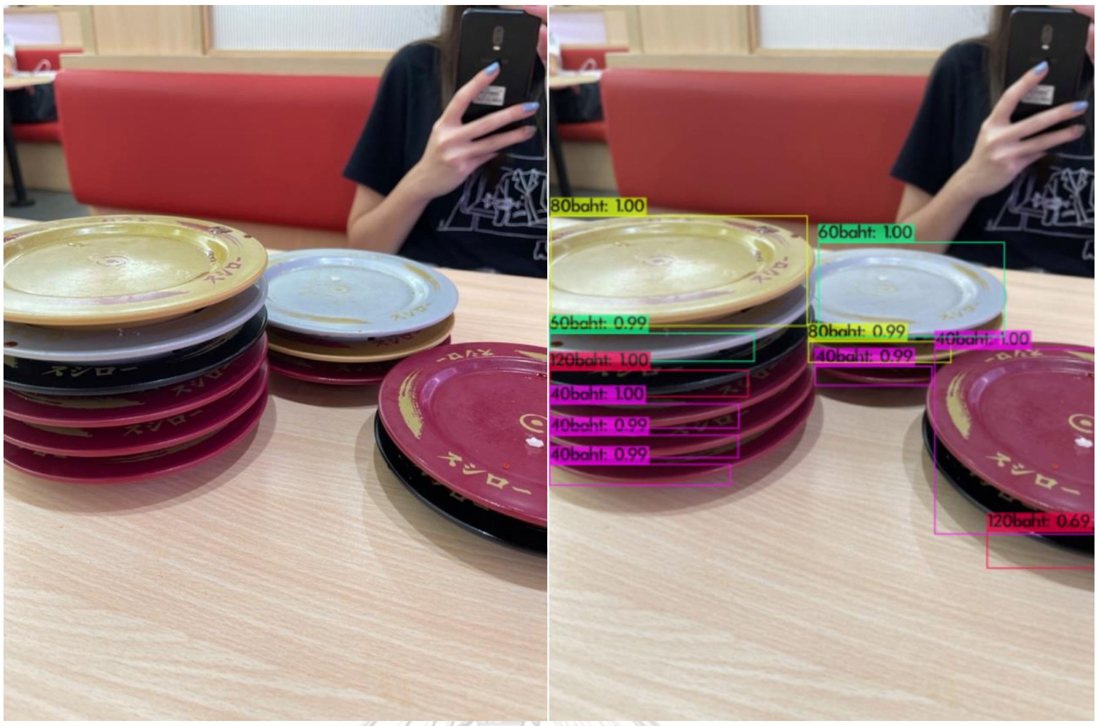


Figure 54: (a) The image in the testset. (b) The result of detection

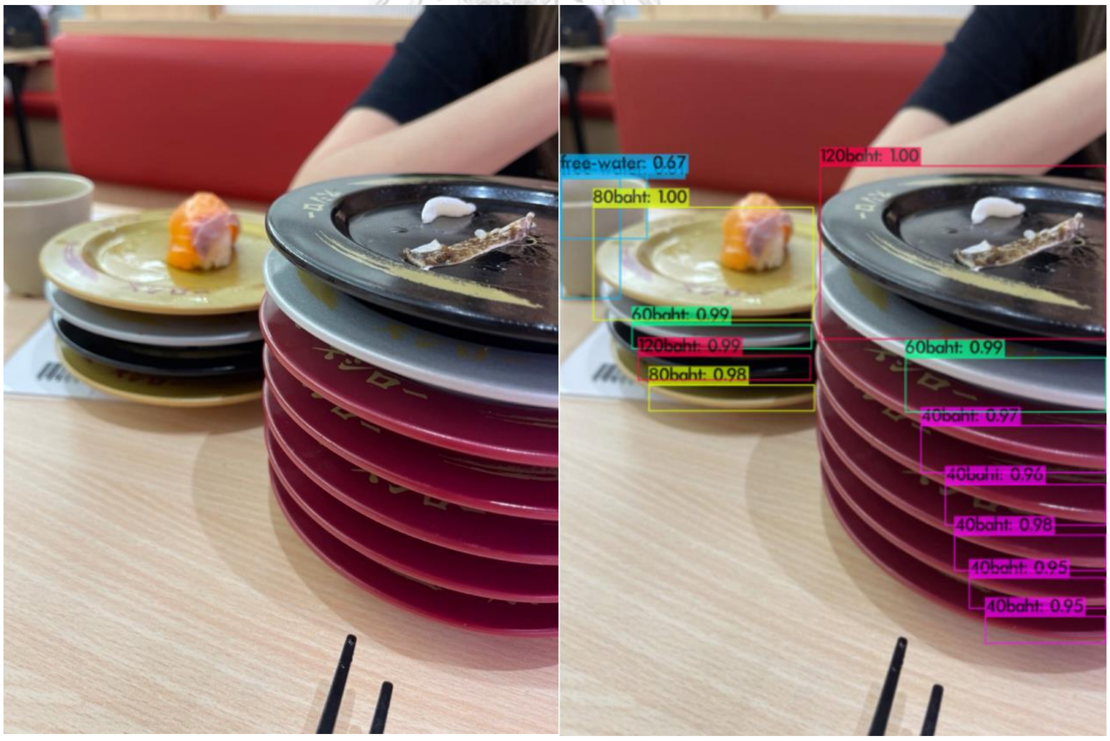


Figure 55: (a) The image in the testset. (b) The result of detection

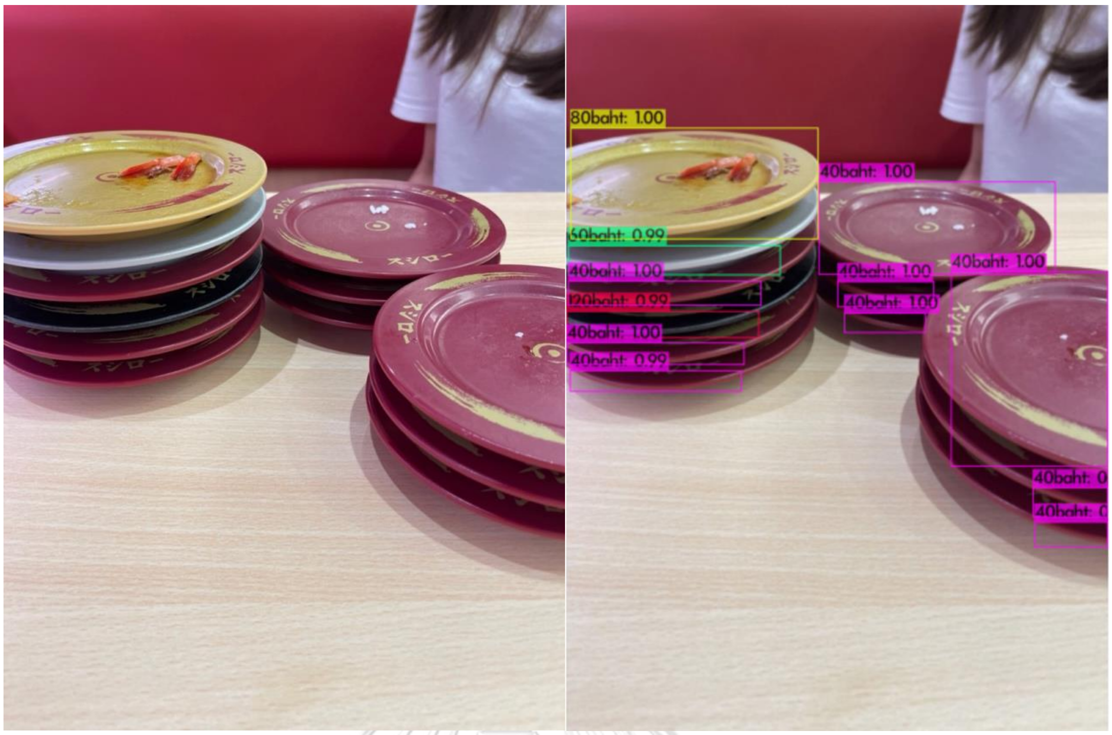


Figure 56: (a) The image in the testset. (b) The result of detection

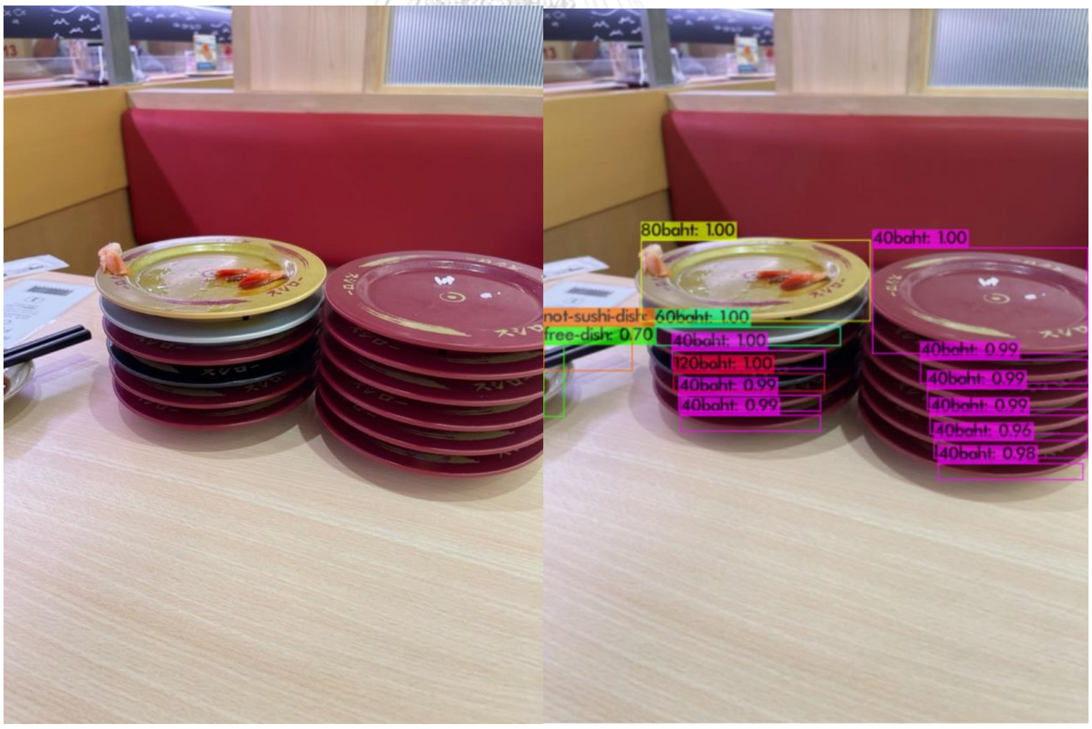


Figure 57: (a) The image in the testset. (b) The result of detection

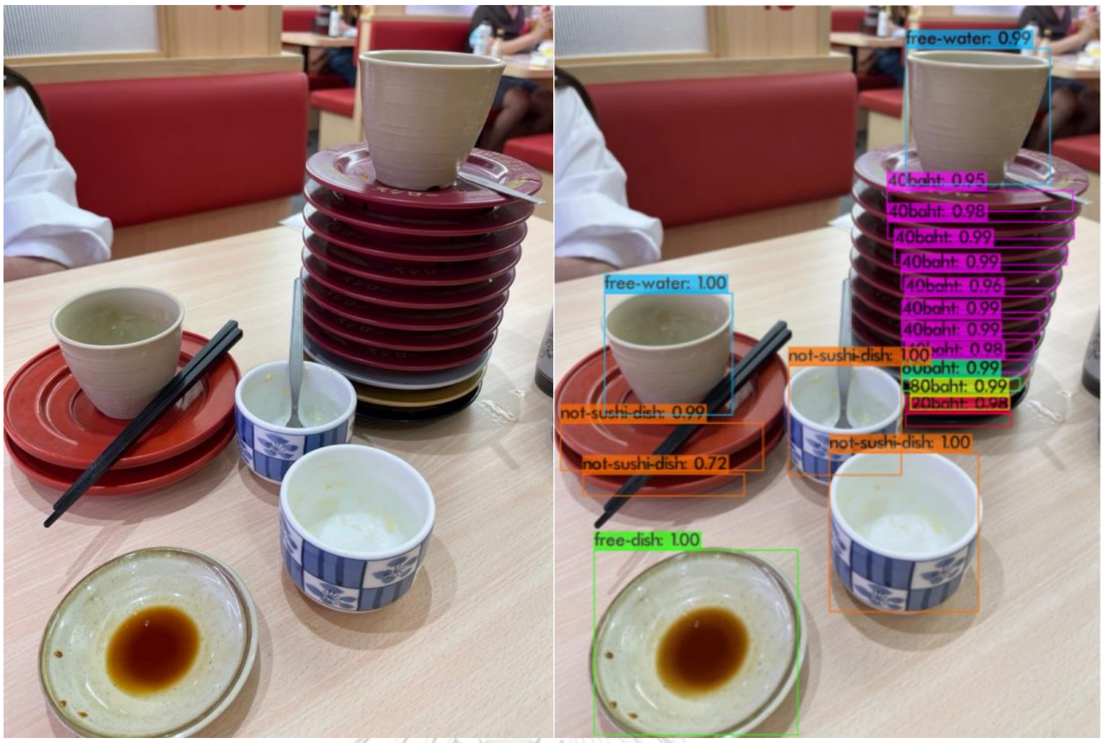


Figure 58: (a) The image in the testset. (b) The result of detection

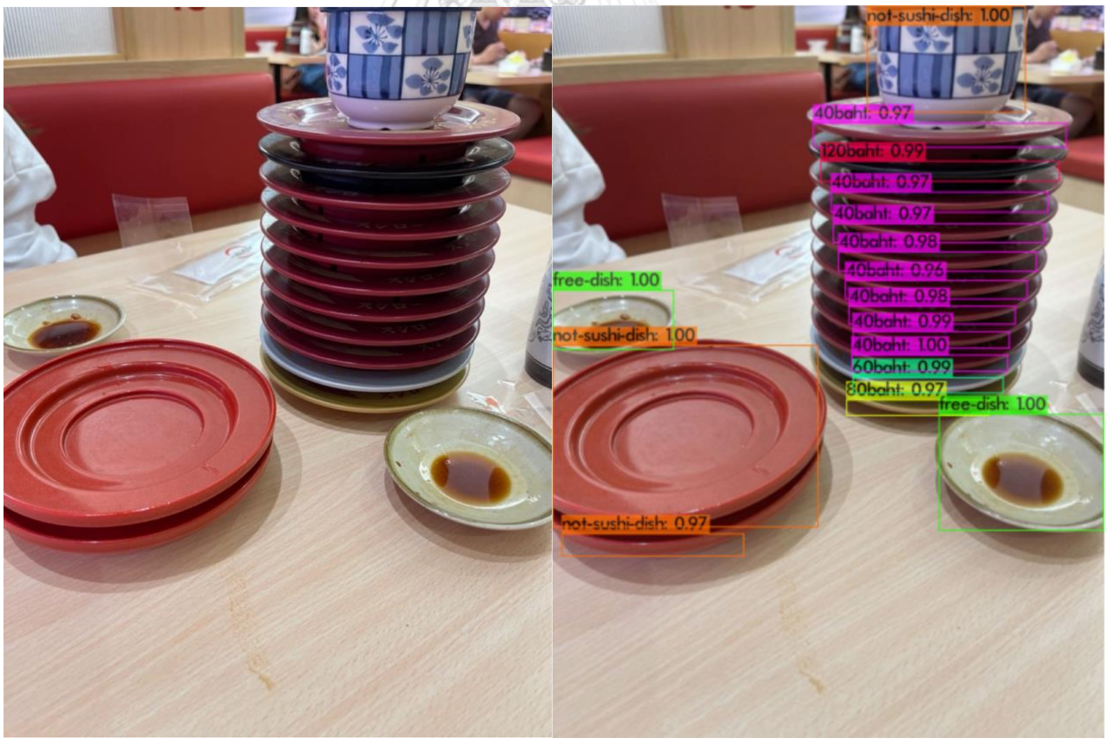


Figure 59: (a) The image in the testset. (b) The result of detection

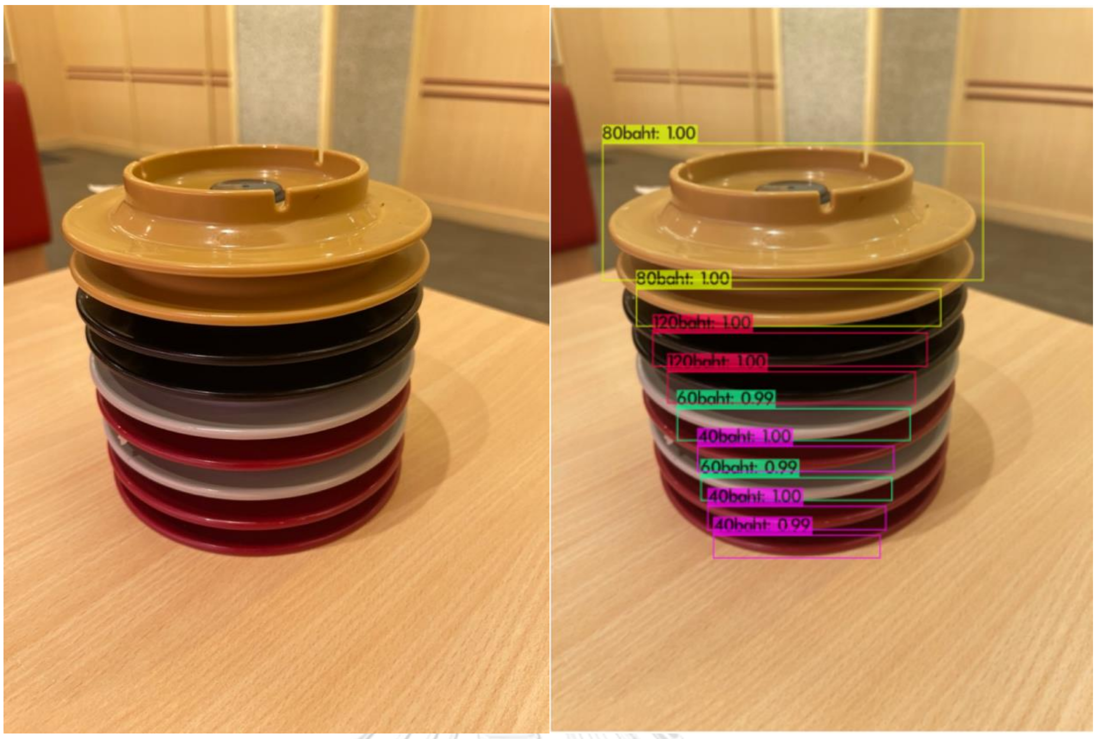


Figure 60: (a) The image in the testset. (b) The result of detection

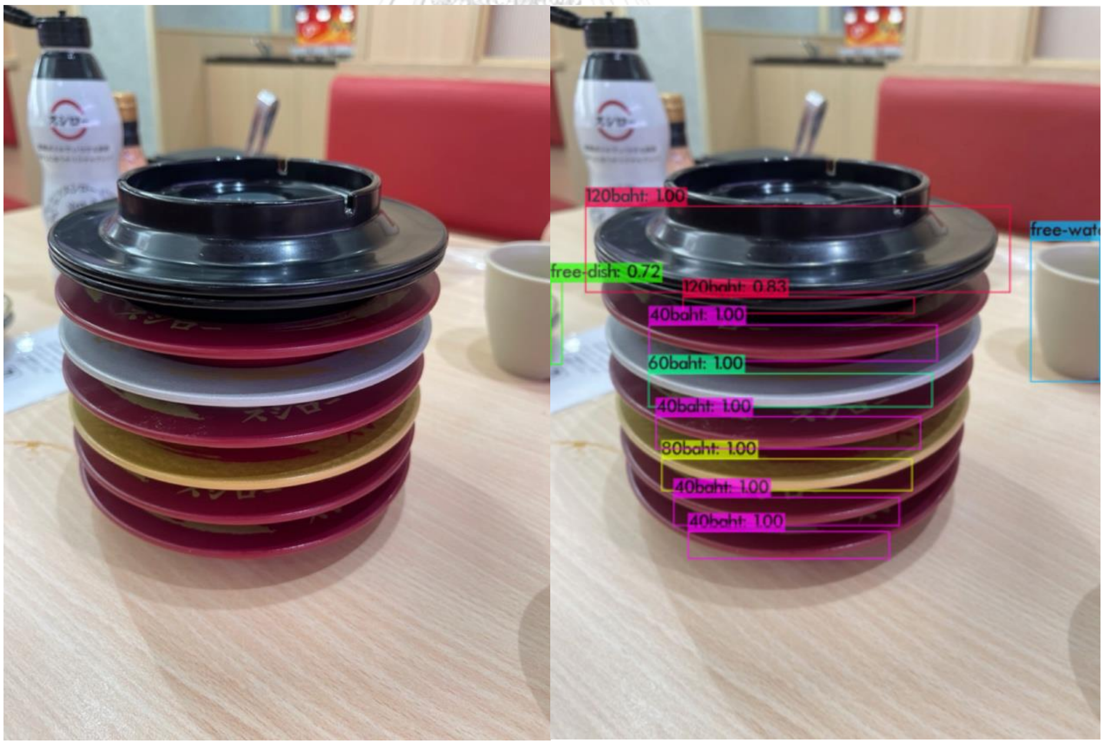


Figure 61: (a) The image in the testset. (b) The result of detection

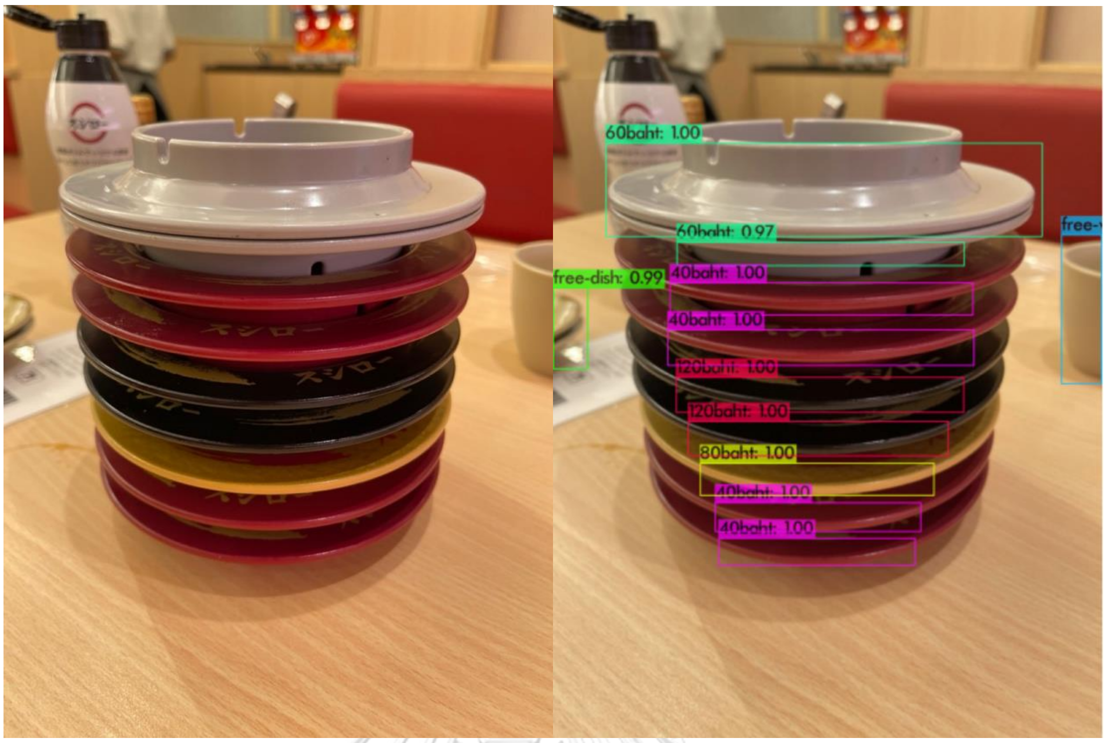


Figure 62: (a) The image in the testset. (b) The result of detection

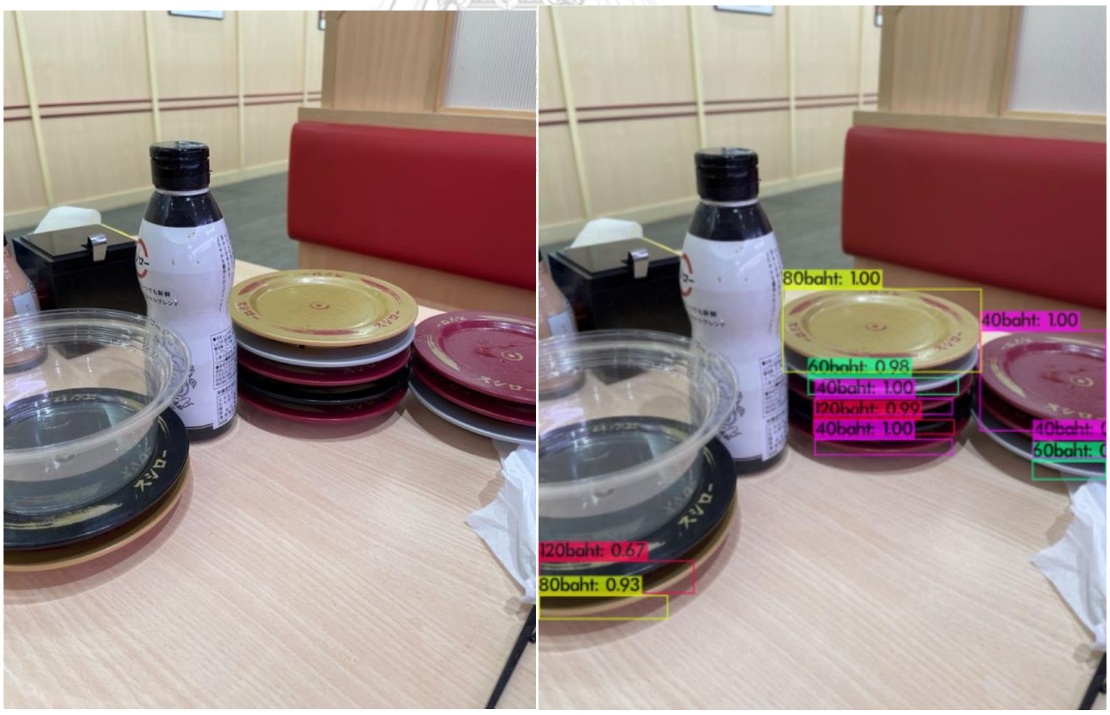


Figure 63: (a) The image in the testset. (b) The result of detection

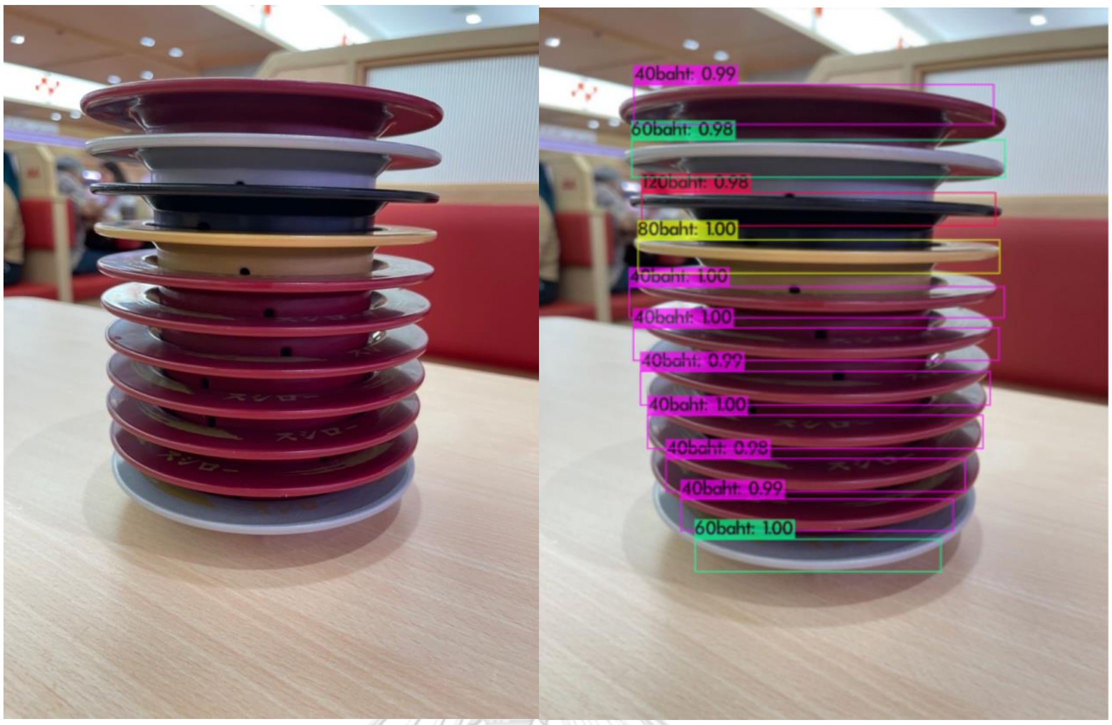


Figure 64: (a) The image in the testset. (b) The result of detection

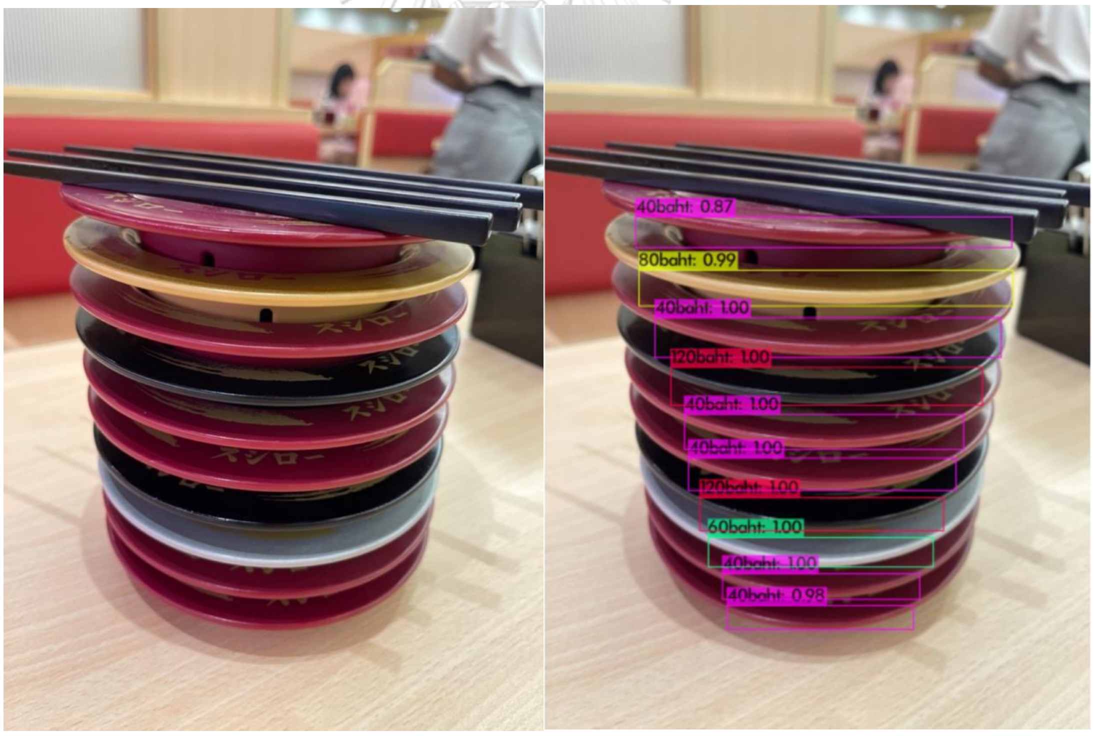


Figure 65: (a) The image in the testset. (b) The result of detection

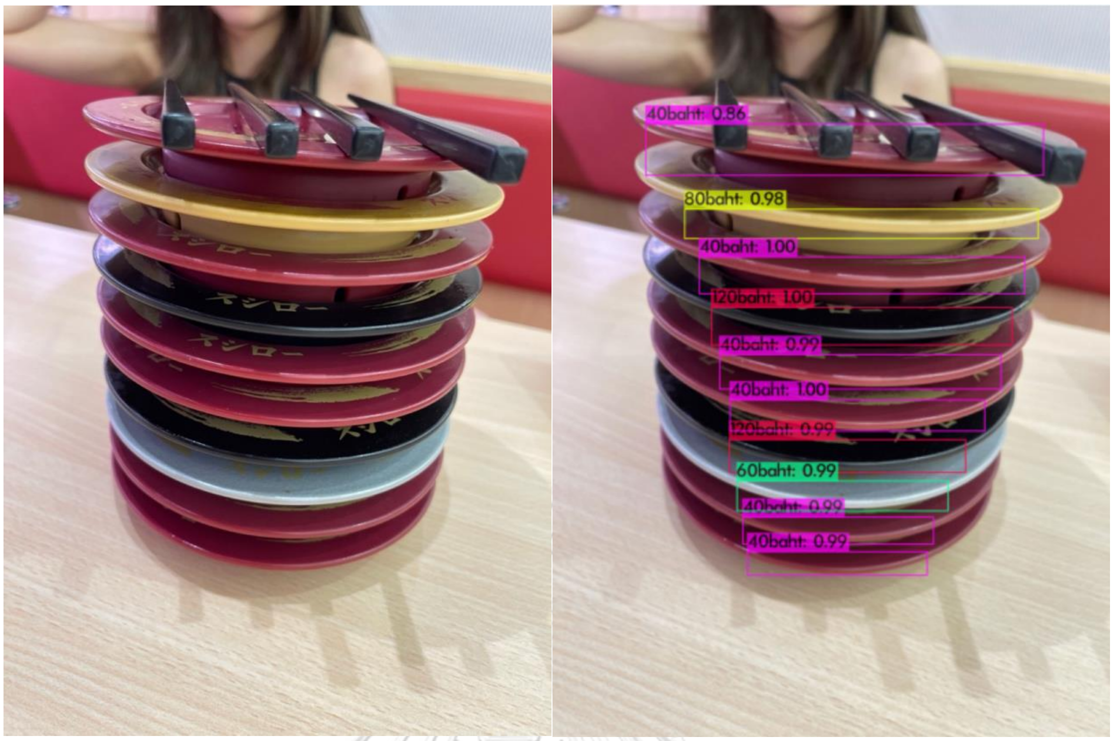


Figure 66: (a) The image in the testset. (b) The result of detection

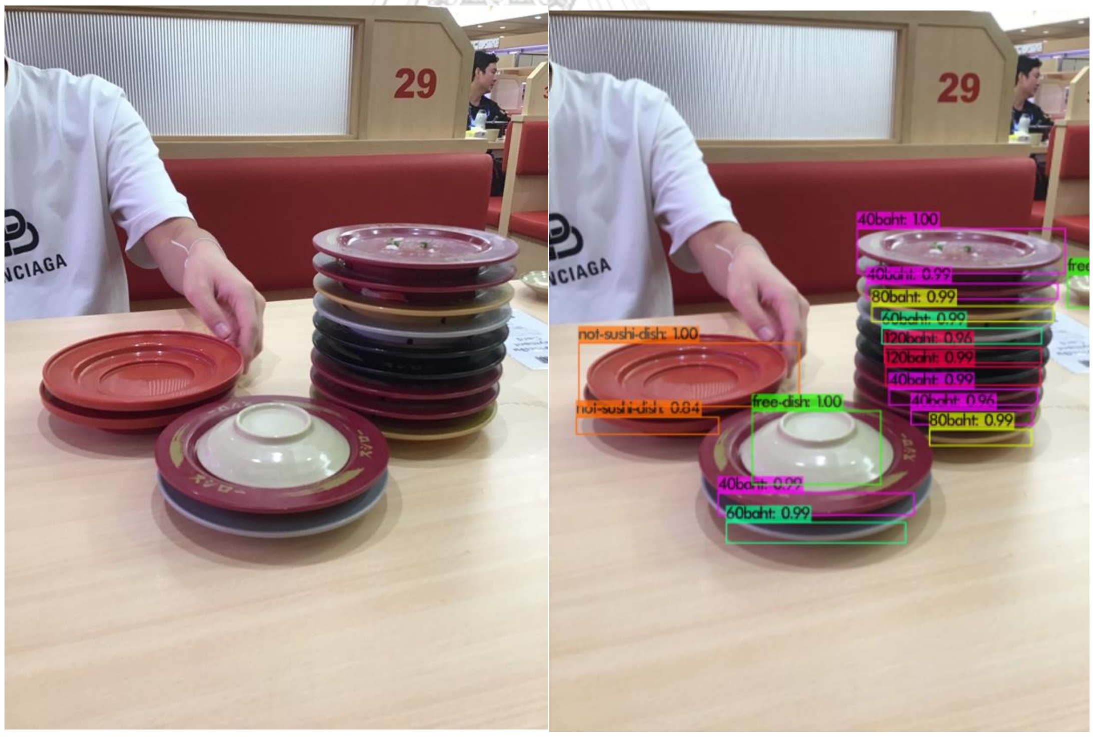


Figure 67: (a) The image in the testset. (b) The result of detection

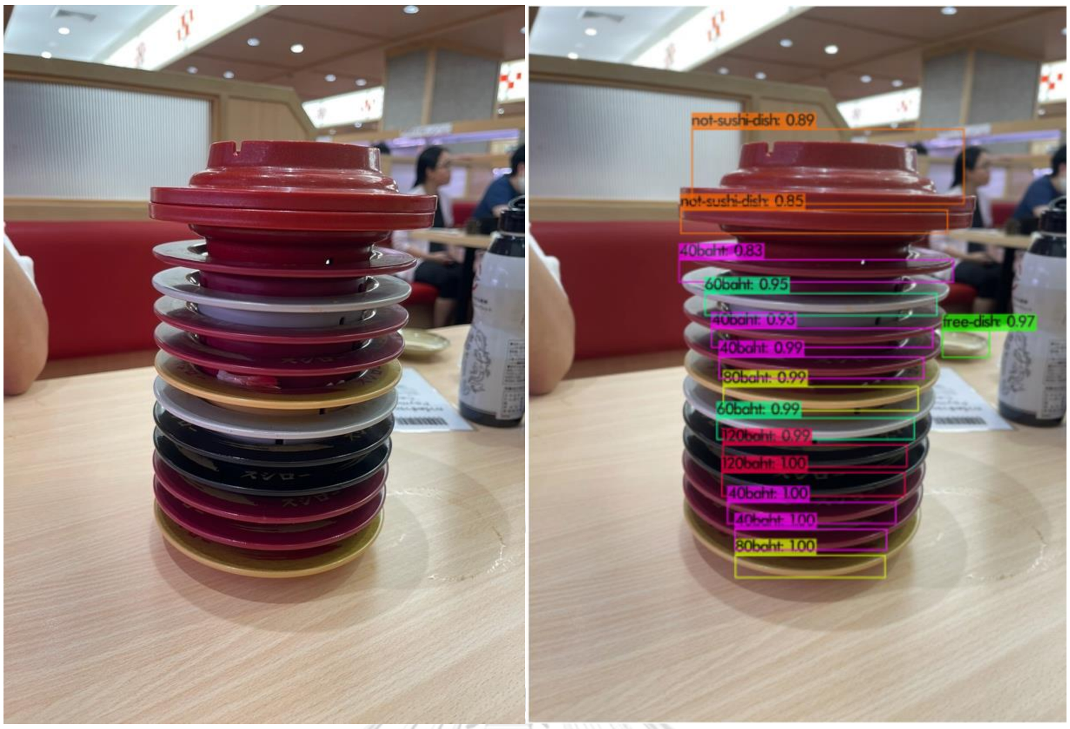


Figure 68: (a) The image in the testset. (b) The result of detection

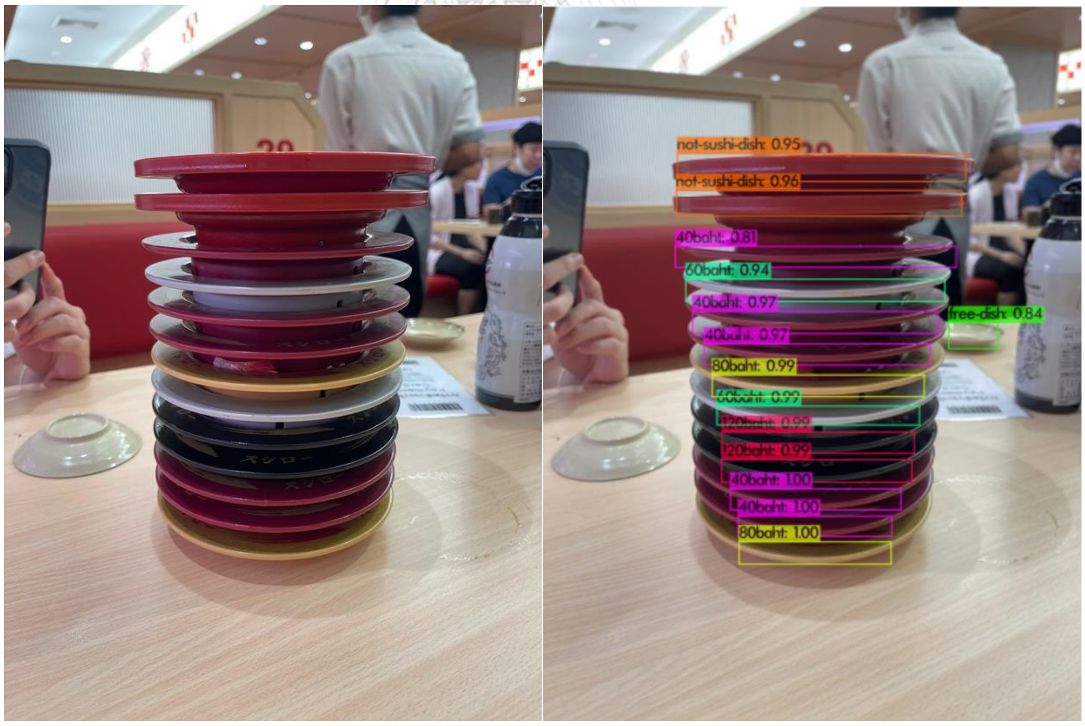


Figure 69: (a) The image in the testset. (b) The result of detection

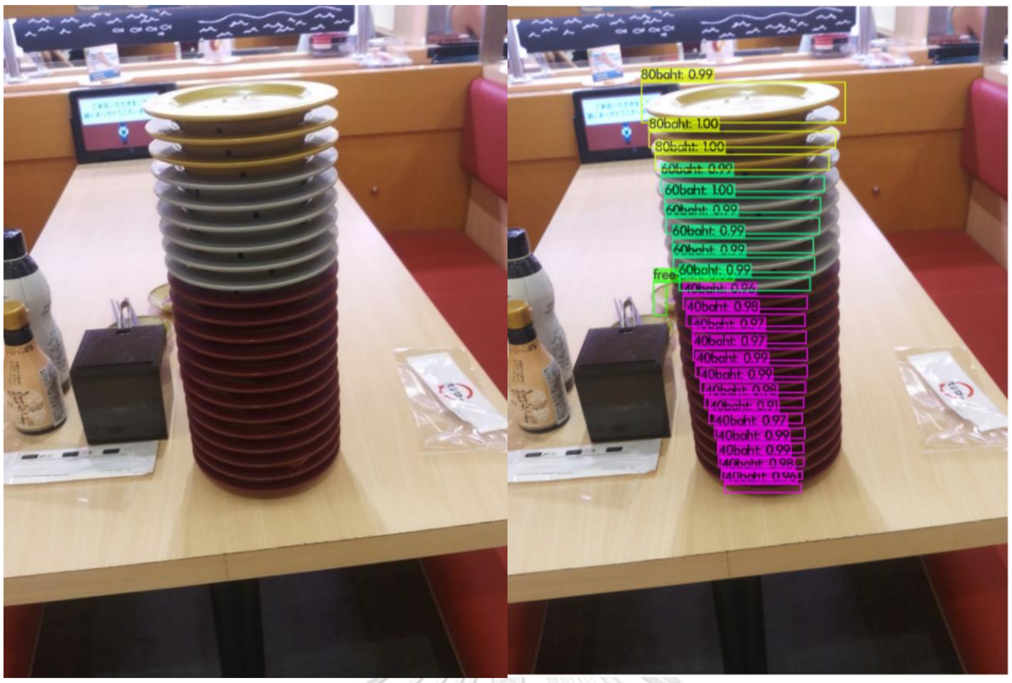


Figure 70: (a) The image in the testset. (b) The result of detection

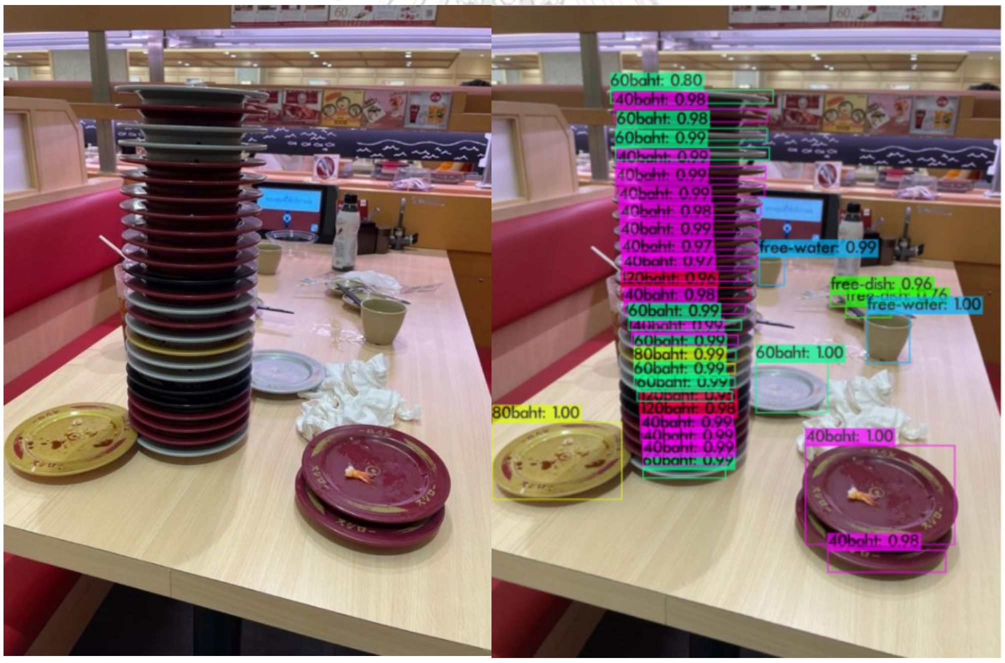


Figure 71: (a) The image in the testset (b) The result of detection

REFERENCES

1. *Wongnai.com*. 2022 [cited 2022 09]; Available from: https://www.wongnai.com/news/sushiro. .
2. *Chill Chill Japan*. 2022 [cited 2022 26]; Available from: https://chillchilljapan.com/5-valuable-conveyor-belt-sushi-franchise/. .
3. *nanareview.com*. 2022 [cited 2022 09]; Available from: https://www.nanareview.com/content/10820/sushiplus-by-sushi-express.
4. *Object Detection*. 2022 [cited 2022 22 Feb]; Available from: https://en.wikipedia.org/wiki/Object_detection.
5. *a guide to two stage object detection r-cnn fpn mask-r-cnn and more*. 2022 [cited 2022 22 Feb]; Available from: https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e168438c.
6. *object recognition with deep learning.* 2022 [cited 2022 22]; Available from: :https://machinelearningmastery.com/object-recognition-with-deep-learning.
7. *spring-boot*. 2022 [cited 2022 15 Feb]; Available from: https://spring.io/projects/spring-boot.
8. *OpenCV*. 2022 [cited 2022 10]; Available from: https://opencv.org/.
9. Kim, J.a., J.Y. Sung, and S.h. Park. *Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition*. in *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. 2020.
10. Bochkovskiy, A., C.-Y. Wang, and H.-Y.M. Liao, *YOLOv4: Optimal Speed and Accuracy of Object Detection.* ArXiv, 2020. **abs/2004.10934**.
11. *The most accurate real-time neural network on MS COCO dataset.* 2022 [cited 2022 9]; Available from: https://alexeyab84.medium.com/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe#:~:text=Neural%20networks%20comparison&text=YOLOv4%20achieves%2043.5%25%20AP%20%2F%2065.7,with%208%E2%80%9316%20GB%20VRAM.
12. Kumar, S., et al. *Object tracking and counting in a zone using YOLOv4,*

DeepSORT and TensorFlow. in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. 2021.

13. Oh, Y., S. Son, and G. Sim. *Sushi Dish - Object detection and classification from real images*. 2022; Available from: https://arxiv.org/abs/1709.00751.
14. *LabelImg is a graphical image annotation tool and label object bounding boxes in images*. 2022; Available from: https://github.com/tzutalin/labelImg.
15. *spring-boot*. 2022 [cited 2022 22 Feb]; Available from: : https://javasterling.com/spring-boot/spring-boot-vs-django/.
16. *Preview macOS*. 2022; Available from: https://en.wikipedia.org/wiki/Preview_(macOS).
17. *Train a custom yolov4 object detector using google colab*. 2022; Available from: https://medium.com/analytics-vidhya/train-a-custom-yolov4-object-detector-using-google-colab-61a659d4868.
18. *YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)*. 2022; Available from: https://github.com/AlexeyAB/darknet.
19. *Custom object detection in the browser using TensorFlow.js*. 2022 [cited 2022 9]; Available from: https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html.

VITA

NAME	Rangrak Maitriboriruks
DATE OF BIRTH	16 September 1997
PLACE OF BIRTH	THAI
INSTITUTIONS ATTENDED	B.Sc. (Computer Science) 2nd class honor, Thammasat University M.Sc. (Computer Engineer) Chulalongkorn University
HOME ADDRESS	222/63 Nawamin 42 Alley Grandio Village Nawamin Rd. Khlong Kum District ,Bangkok 10240