LAB: Final Project

LAB: Management of Expiration Dates and Inventory in Unmanned Stores

- Class : Deep Learning Image Processing by prof. Youngkeun Kim
- Date : 2024-June-24
- Author : Gyeonheal An 21900416, Taegeon Han 21900793
- GitHub : GitHub link -GyeonhealAn, Github link - TaegeonHan
- Demo Video : Youtube Link 1, Youtube Link 2

I. Introduction

In this lab, we are required to solve problems that commonly occur in unmanned stores. One of the main problems with unmanned stores is that items don't have price tags, so customers don't know the prices. Additionally, there are cases where expired products remain on the shelves, making it impossible to eat food. Conversely, the items you need may not be on the shelves, which can cause inconvenience to customers. We need to know the expiration date of the products and inform the store manager if the product is available, and design a system to measure the price in case there is no price tag.

To solve this problem, we need to design a comprehensive system that can determine product expiration dates and monitor inventory on shelves. The system notifies store managers when products are expired or out of stock. Additionally, we will design a price measurement system that allows customers to know the price information for all products to solve the problem of not having a price tag.


(a) Out of stock (b) No price tag (c) Past the expiration date

Figure 1. Problems in unmanned store

1. Requirement

Hardware

- Webcam

Software Installation

- CUDA 11.8
- cudatoolkit 11.8.89
- Python 3.8.18
- Pytorch 2.1.2

- Torchvision 0.16.2
- YOLO v8

Set up

Anaconda Installation

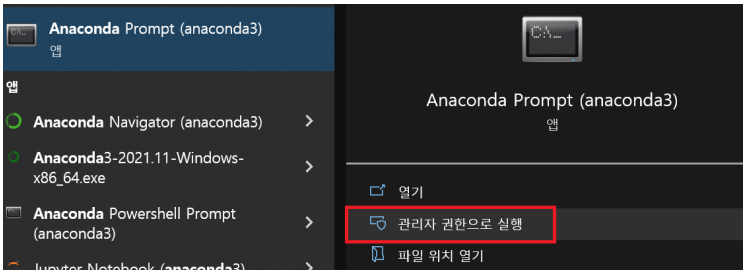
Follow the link below and install it

Link: [How to install Anaconda]

Create virtual environment for Python 3.9

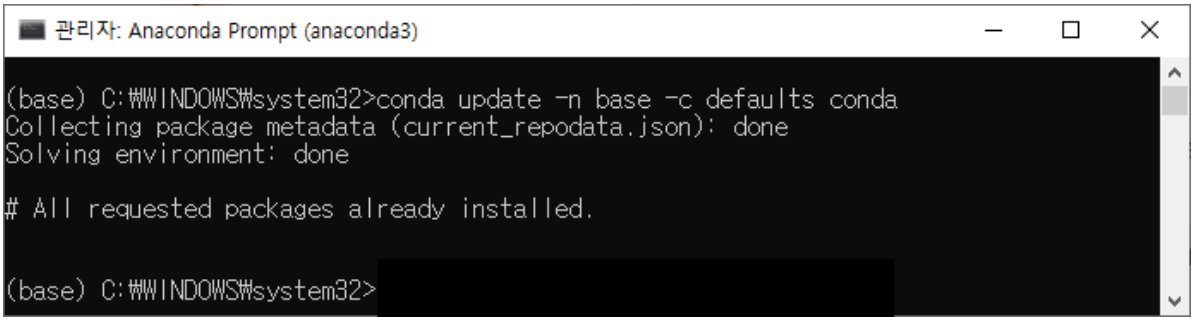
Python is already installed by installing Anaconda. But, we will make a virtual environment for a specific Python version.

- Open Anaconda Prompt(admin mode)



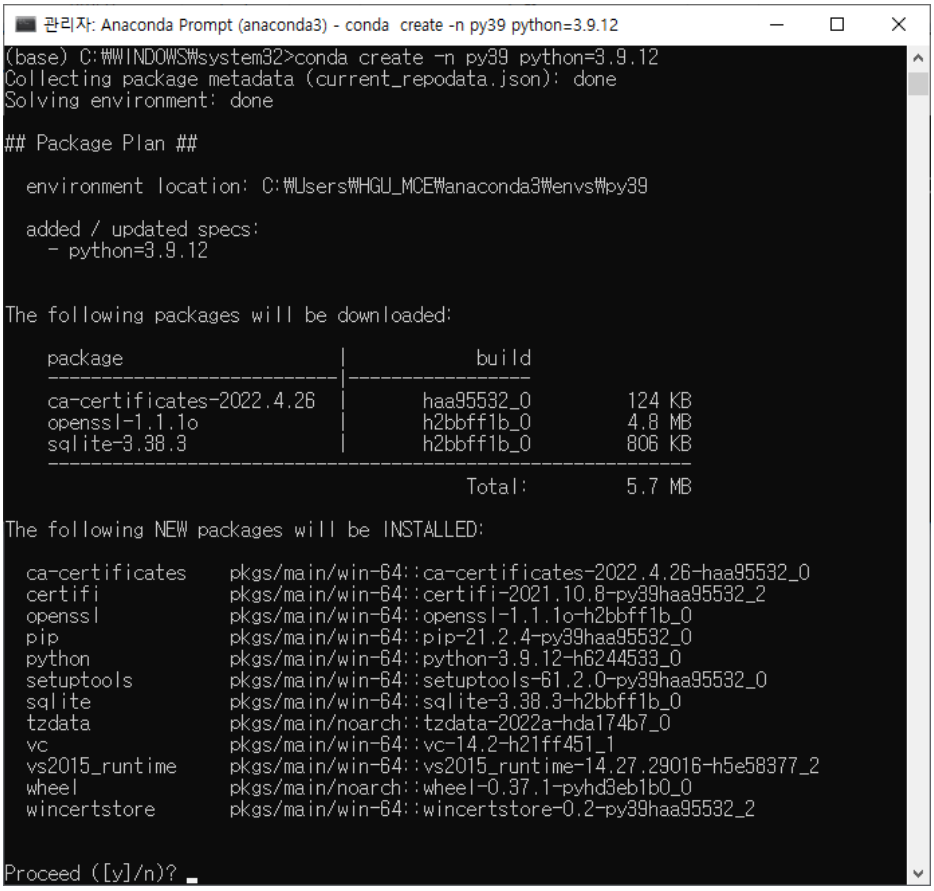
- First, update conda

```
conda update -n base -c defaults conda
```



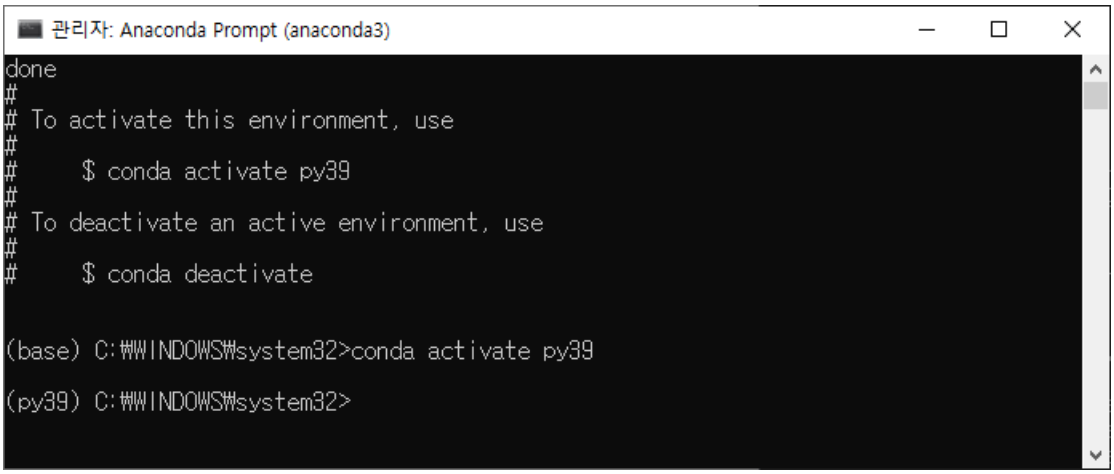
- Then, Create virtual environment for Python 3.9. Name the $ENV as `py39`. If you are in base, enter `conda activate py39`

```
conda create -n py39 python=3.9.12
```



- After installation, activate the newly created environment

```
conda activate py39
```



Libs Installation

Install Library that we need to conduct project

```
conda activate py39
conda install -c anaconda seaborn jupyter
pip install opencv-python
pip install smtplib
pip install mediapipe
```

Visual studio code Installation

Follow the link below and install it

Link: How to install VS Code

CUDA 11.8 Installation

Follow the link below and install it

Link: How to install CUDA

Yolov8 Installation

Follow the link below and install it

Link: How to install Yolov8

DarkLabel Installation

Follow the link below and install it

https://github.com/darkpgmr/DarkLabel

SMTP setting

To send G-mail using Python code, you need to enable IMAP. From G-mail settings, you must follow the below to send an e-mail by Python.

2. Dataset

We captured convenience store food items and obtained approximately 7,000 frames. These frames were used to train the model, utilizing the Yolov8 pretrained model to recognize different types of food. Each food item needs to be recognized separately, and they were categorized into five classes: triangle kimbap, milk, coffee, yogurt, and sandwich.

[Dataset link 1](#) or [Dataset link 2](#)

Figure 2. Convenience Store Food Items Used for Training Dataset

II. Procedure

1. Training Process

1.1. Labeling

We went to a school convenience store and took videos of the foods we wanted. From there, several frames were extracted and each food was labeled. At this time, the DarkLabel program was used, which can be used to label desired images or videos after execution.

The classes were set to triangular gimbap, milk, coffee, yogurt, and sandwich from 0 to 4, respectively.

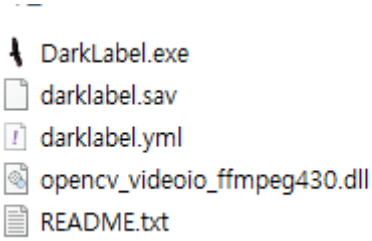


Figure 3. Labeling Examples

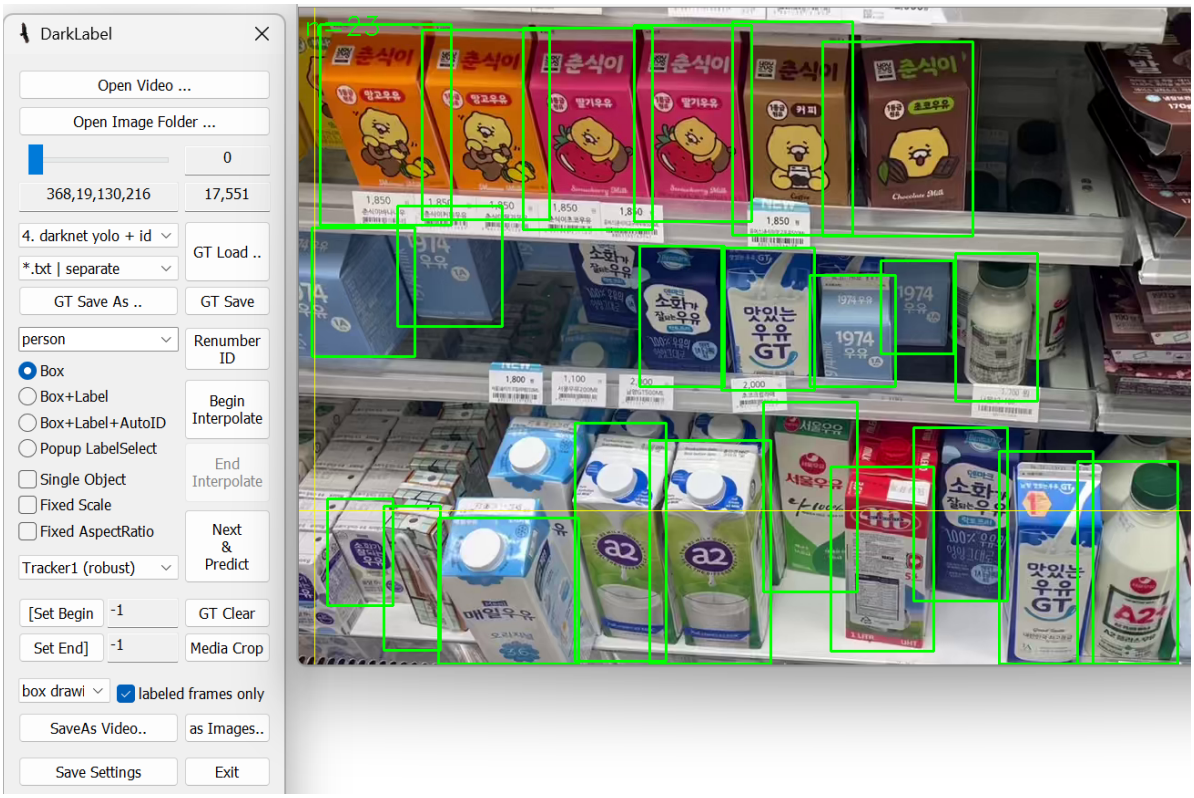


Figure 4. Labeling using DarkLabel

As shown in Figure 4, recorded videos are used for labeling to determine each class for each item. After labeling is complete, save the image file and txt file. If you open the txt file, you can see that unnecessary values such as '-1' are in column 2. Therefore, it was removed using the matlab code below. We can see the Matlab results in Figure 5.

```
inputDir = 'file_path';

for fileNum = 0:3715
    % Construct the input file name with leading zeros
    fileName = sprintf('%s%08d.txt', inputDir, fileNum);

    % Load the data
    if isfile(fileName)
        data = load(fileName);

        % Remove the second column
        data(:, 2) = [];

        % Open the same file for writing (overwrite)
        fileID = fopen(fileName, 'w');

        % Write the modified data to the file with the desired format
        [nrows, ncols] = size(data);
        for row = 1:nrows
            fprintf(fileID, '%d ', data(row, 1)); % Write the first column as an
integer
            fprintf(fileID, '%.6f ', data(row, 2:end)); % Write the rest as
floating point numbers with 6 decimal places
            fprintf(fileID, '\n');
        end

        % Close the file
        fclose(fileID);
```

```
    else
        fprintf('File %s does not exist.\n', fileName);
    end
end
```

All txt files contained -1 in column 2, so when we ran the code, -1 was removed from all txt files.

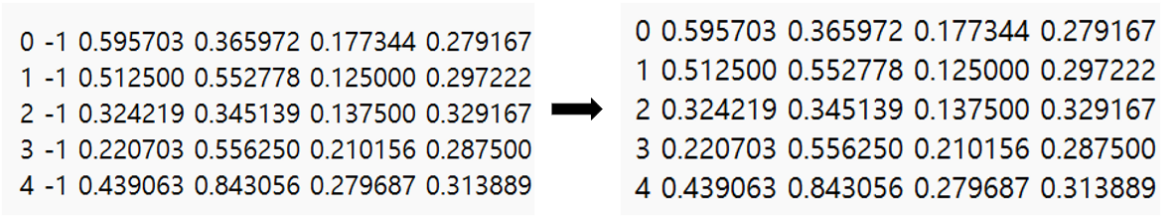


Figure 5. Txt files excluding -1 using Matlab

1.2 Data classification

Before using the YOLO model, the data must be partitioned. The data at this time can be divided into image files and text files. The text file contains class and labeling coordinates.

Figure 6. Separating data image files and text files

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * dataset split code

import os, shutil, random

# preparing the folder structure

full_data_path = 'datasets/food/Archive/milk/'
extension_allowed = '.png'
split_percentage = 90

images_path = 'datasets/food/Archive/milk/images/'
if os.path.exists(images_path):
    shutil.rmtree(images_path)
os.mkdir(images_path)

labels_path = 'datasets/food/Archive/milk/labels/'
if os.path.exists(labels_path):
    shutil.rmtree(labels_path)
os.mkdir(labels_path)
```

```
training_images_path = images_path + 'training/'
validation_images_path = images_path + 'validation/'
training_labels_path = labels_path + 'training/'
validation_labels_path = labels_path + 'validation/'

os.mkdir(training_images_path)
os.mkdir(validation_images_path)
os.mkdir(training_labels_path)
os.mkdir(validation_labels_path)

files = []

ext_len = len(extension_allowed)

for r, d, f in os.walk(full_data_path):
    for file in f:
        if file.endswith(extension_allowed):
            strip = file[0:len(file) - ext_len]
            files.append(strip)

random.shuffle(files)

size = len(files)

split = int(split_percentage * size / 100)

print("copying training data")
for i in range(split):
    strip = files[i]

    image_file = strip + extension_allowed
    src_image = full_data_path + image_file
    shutil.copy(src_image, training_images_path)

    annotation_file = strip + '.txt'
    src_label = full_data_path + annotation_file
    shutil.copy(src_label, training_labels_path)

print("copying validation data")
for i in range(split, size):
    strip = files[i]

    image_file = strip + extension_allowed
    src_image = full_data_path + image_file
    shutil.copy(src_image, validation_images_path)

    annotation_file = strip + '.txt'
    src_label = full_data_path + annotation_file
    shutil.copy(src_label, validation_labels_path)

print("finished")
```

This code creates `images` and `labels` folders, and splits the images and text files into training and validation sets in a 9:1 ratio, storing them accordingly.

After that, create a file to train and a file to verify in the same location as the file above. Since the images file and labels file are separate, so you can create them inside each. We can classify training data and validation data using the reference code.

1.3. Select Model

To recognize labeled foods, we need to find a suitable pre-trained model. Among several YOLO models, YOLOv8 was used in this project considering GPU driver performance. The types of YOLOv8 models can be seen in Figure 5. Among them, YOLOv8s was selected considering precision and training speed.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 7. Model type of YOLOv8

1.4. Train model

First, create a yaml file, check the paths of training data and verification data here, and then specify the name of the class.

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * yaml code for training

train:
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\coffee\images\tr
aining
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\milk\images\trai
ning
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\yogurt\images\tr
aining
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\triangle_kimbap\
images\training
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\sandwich\images\
training
```

```
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\collection\image
s\training
val:
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\coffee\images\va
lidation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\milk\images\vali
dation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\yogurt\images\va
lidation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\triangle_kimbap\
images\validation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\sandwich\images\
validation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\collection\image
s\validation

# number of classes
nc: 5

# class names
names: ['Triangle kimbap', 'Milk', 'Coffee', 'Yogurt', 'Sandwich']
```

Caution: When setting the path for image data in the yaml file using relative paths, an error occurred in the training code as it could not find the specified paths. Therefore, the absolute paths in the code above should be replaced with your own absolute paths to enable training.

After that, we use the selected YOLOv8s model. At this time, you must set the epoch value. A value that is too large increases the risk of overfitting, and a value that is too small increases the risk of underfitting. It is important to train the model by setting appropriate epoch values. Therefore, we trained the model by setting it to 15. After training the model, a best.pt file is created. The best.pt file is a model weight file with optimal training parameter weights. This value can be used to detect trained food images in real-time video.

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * training code

from ultralytics import YOLO

def train():
    # Load a pretrained YOLO model
    model = YOLO('yolov8s.pt')

    # Train the model using the DLIP_Final.yaml dataset for epochs
    results = model.train(data="DLIP_Final.yaml", epochs=30)
```

```
if __name__ == '__main__':
    train()
```

2. Flow Chart

This is a flowchart of this lab. It detects food or checks its shelf life, and can indicate its price..

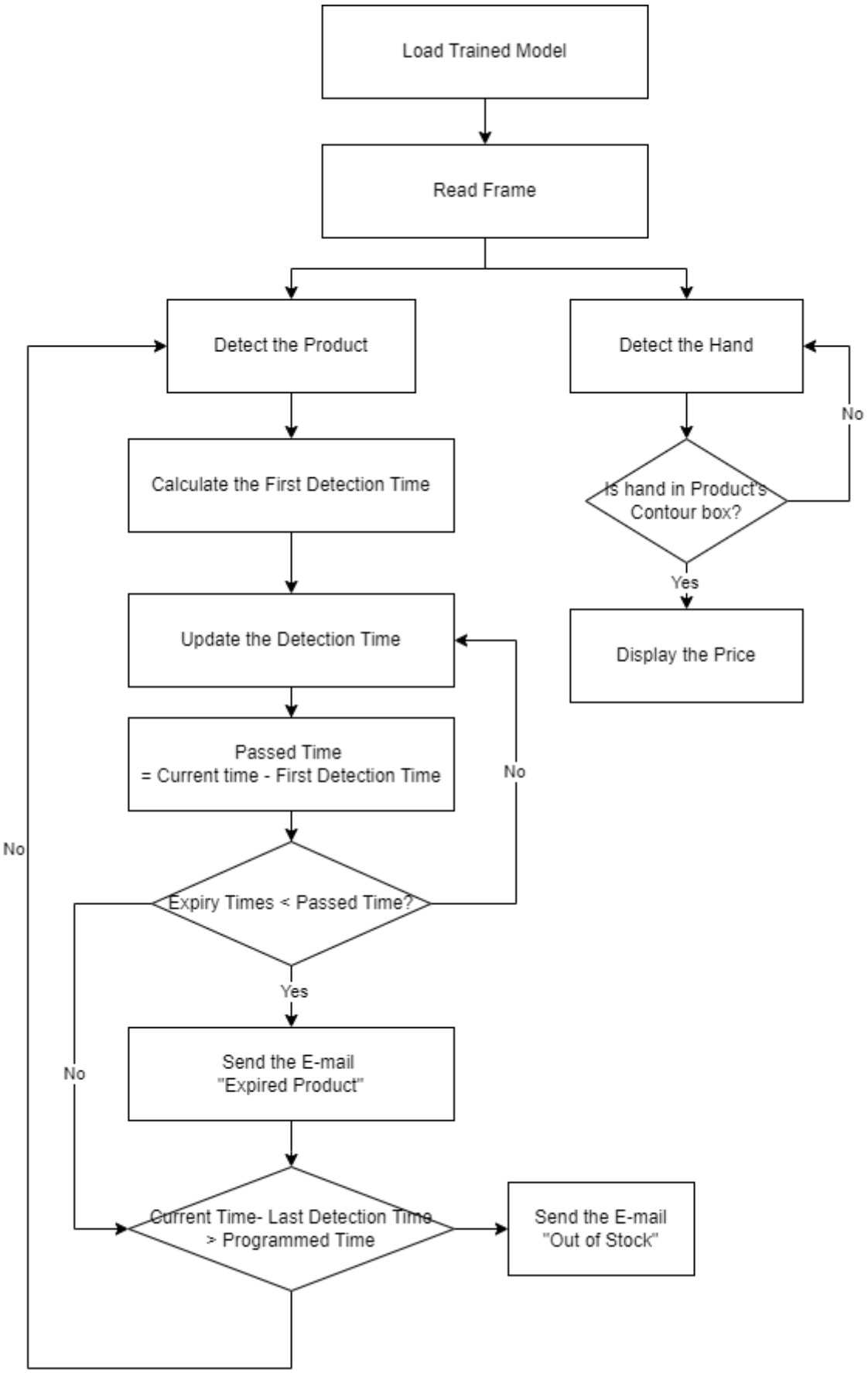


Figure 8. Flow chart of final lab

Expiration Date Management

1. First, detect objects using the webcam while simultaneously recording the initial detection time.
2. Assign the current time to the initially detected class and set the email flags for that class to False.

3. Calculate the elapsed time from the initial detection time and compare it to the expiration date.
4. If the expiration date has passed, send an email notification indicating that the expiration date has passed and set the email flag to True.
5. If the expiration date has not passed, update the elapsed time.

Inventory Management

1. If a class item is not detected for the programmed duration (5 seconds), send an email notification indicating the item is out of stock and set the email flag to True.
2. Prevent duplicate email notifications.

Hand Detection

1. Detect the hand.
2. If the detected hand is within the contour box, display the price.

2.1. Load trained model & Detect the Product

Five types of food were learned. Milk, yogurt, triangle kimbap, sandwich, and coffee. It was made into a wood rack modeled after the shape of a refrigerator so that items can be placed there. Five objects were placed on top of it, and a webcam was installed at a certain distance so that all objects could be seen on the screen. A image of this is shown in Figure 8. If you look at the image, we can see that they recognize foods well. This means that the training went well for each class during the training process.

Figure 9. Detected foods

```
def main():
    model = YOLO('YOLOv8s_30.pt')

    webcam = cv.VideoCapture("test_video.mp4")
    # webcam = cv.VideoCapture(1)

    if not webcam.isOpened():
        print("Could not open webcam")
        return None
```

```
    while webcam.isOpened():
        frame, display_frame, results = process_frame(webcam, model)
        if frame is None:
            break

        current_time = time.time()                                    #
Updating Current Time
```

2.2. Calculate expiration dates

The expiration date time was set for each class. In reality, it takes several days, but since the experiment must be conducted within the given time, the expiration date is set in seconds for each class. Triangular kimbap was set at 5 seconds, milk was set at 10 seconds, yogurt was set at 15 seconds, sandwich was set at 20 seconds, and coffee was set at 25 seconds. The system stores the initial detection time for each class and continuously monitors it to calculate the time elapsed since the first detection. If the elapsed time exceeds the preset expiration time, a message is displayed on the food indicating that the food has expired. Figure 9 shows foods that have passed their expiration date. The name and confidence score of the class appear along with the message "[class name] expired".

Figure 10. Foods that have past expiration dates

```
def update_detection_times(results, current_time, display_frame): # Calculate
Passed Time and Determine if the expiration date has passed
    detected_classes = set()
    for result in results[0].boxes:
        class_id = int(result.cls)
        class_name = class_names[class_id]
        detected_classes.add(class_name)

        x1, y1, x2, y2 = map(int, result.xyxy[0])
        if class_name not in first_detection:                         # When it is
detected first time --> Add the class
            first_detection[class_name] = current_time                # Indexing
the first detection time for each classes
            expired_email_flag[class_name] = False                    #
Initializing the flag
```

```
            stock_email_flag[class_name] = False
        last_detection[class_name] = current_time
        passed_time = current_time - first_detection[class_name]    #
Calculating passed time

        if passed_time > expiry_times[class_name]:                  # When the
product passed the expired date
            cv.putText(display_frame, f"{class_name} expired", (x1, y1 - 30),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv.LINE_AA)
            if not expired_email_flag[class_name]:                  # Sending E-
mail
                remaining_times = []                                # Store the
remain times for product which not passed the expired date
                for other_class in class_names:                     # whole
classes
                    if other_class in first_detection and other_class !=
class_name:    # A class other than the currently expired product class that has
already been detected
                        remaining_time = expiry_times[other_class] -
(current_time - first_detection[other_class])  # Calculating Remain time
                        remaining_times.append(f"{other_class}:
{remaining_time:.2f} seconds remaining")
                msg_body = f"The {class_name} is expired.\n\nRemaining times for
other items:\n" + "\n".join(remaining_times)   # Message
                send_email(f"{class_name} is expired", msg_body)
                expired_email_flag[class_name] = True

    check_stock_empty(detected_classes, current_time)
```

2.3. Inventory Management

If at least one of the five items in a class is not detected for a certain period of time, an email is sent to the store owner indicating that the item is out of stock, and the `stock_email_flag` value is set to True to prevent duplicate emails.

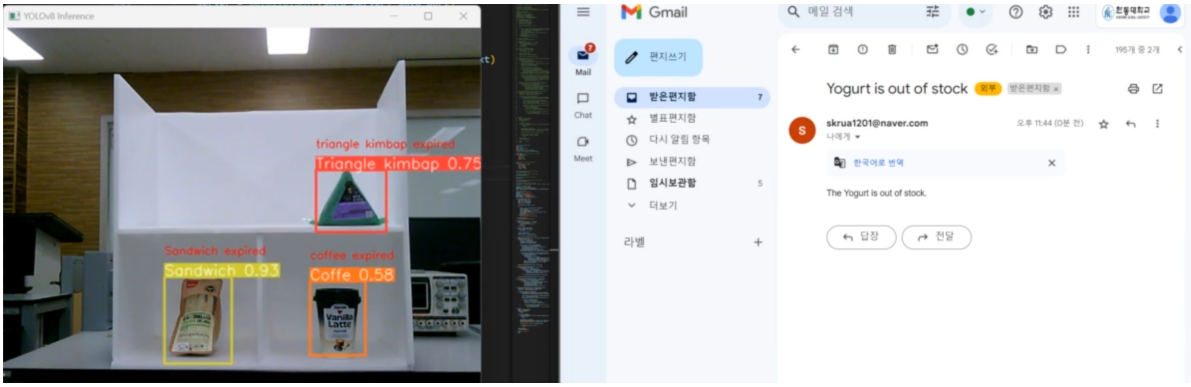


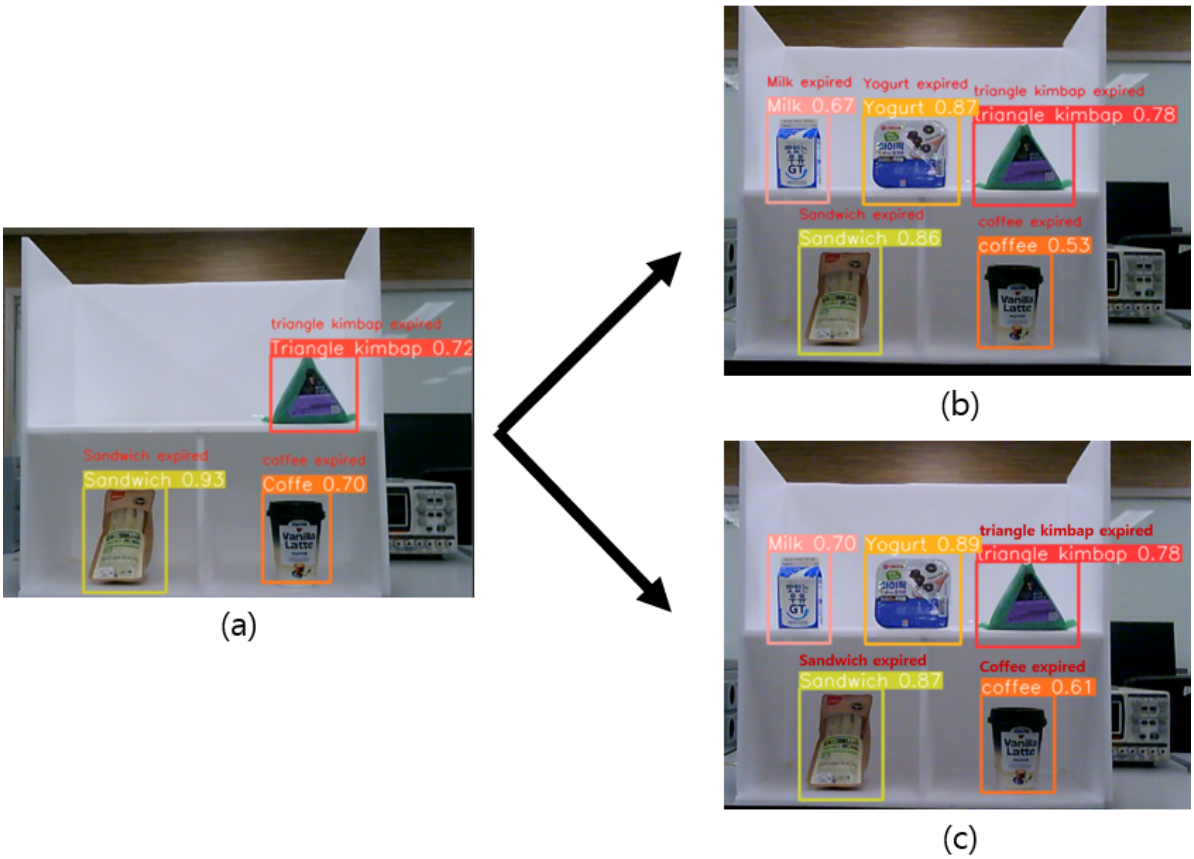
Figure 11. Out of stock situation

```
def check_stock_empty(detected_classes, current_time):
    for class_name in class_names:
        if class_name not in detected_classes:
            if class_name in last_detection:
                if current_time - last_detection[class_name] > 5 and not
stock_email_flag.get(class_name, False):       # If it is not detected over
programmed times
                    send_email(f"{class_name} is out of stock", f"The
{class_name} is out of stock.")
                    stock_email_flag[class_name] = True             # E-mail
flag not to send E-mail duplicate
            else:
                last_detection[class_name] = current_time           # Update the
last detection time

    reset_detection_times(detected_classes, current_time)
```

2.4. Food status judgment (Detection Time Reset Method)

Considering a scenario where food could be temporarily removed and put back in, the re-introduction time was set to a maximum of 5 seconds. That is, if the food is returned within 5 seconds after it disappears, the system considers it the same food. Taking the image as an example, if you take the food in an expired state as shown in Figure 9, and then return the food within 5 seconds as shown in Figure 10(a), the message that the expiration date has passed is maintained, as shown in Figure 10(b). If it is placed back after 5 seconds, it is assumed that new food has been added, the expiration date is reset, and the message disappears, as shown in Figure 10(c). In this case, the message regarding expiration will appear only when the expiration date has passed again.



(a) (b) (c)

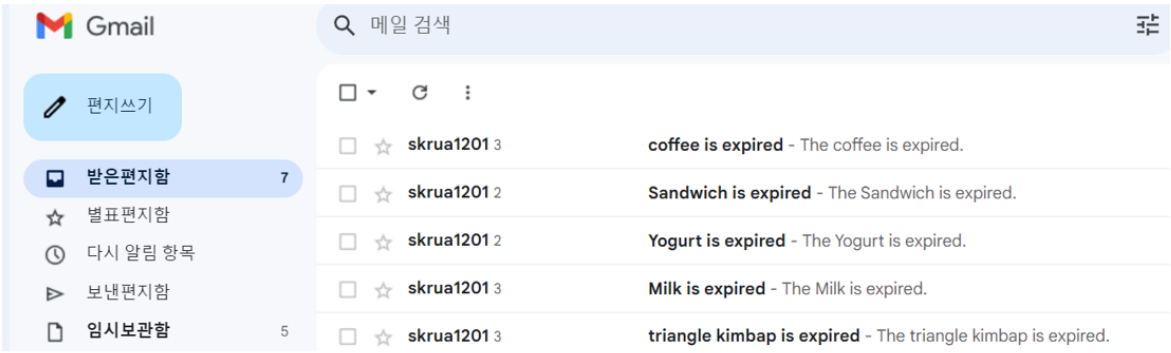
Figure 12. Two cases based on the time between taking food and putting it back

```
def reset_detection_times(detected_classes, current_time):
    for class_name in list(first_detection.keys()):
        if class_name not in detected_classes:
            if current_time - last_detection[class_name] > 5:        # When it is
not detected over programmed times
                first_detection.pop(class_name)                      # Clear all
the product index and flags
                last_detection.pop(class_name)
                expired_email_flag.pop(class_name)
                stock_email_flag.pop(class_name, None)
```

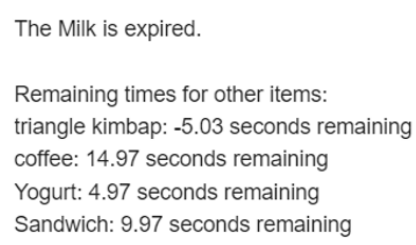
2.5. Send e-mail

There are two situations in which you might send an email to your manager's email address. First, an email notification is sent when the expiration date of the food expires. For example, as shown in Figure 9, when the expiration date is exceeded, a total of 5 emails are sent, one for each expired item, as shown in Figure 11(a). By clicking on the email, you can check the time remaining until the expiration date of the remaining foods. Or, it shows how much time has passed for expired food since the expiration date. In this example, the expiration date of the milk has expired, and since the expiration date of milk is the second shortest, if you look at Figure 11(b), you can see that the time of triangle kimbap, which has the shortest expiration date, is negative, indicating that the expiration date has already passed.

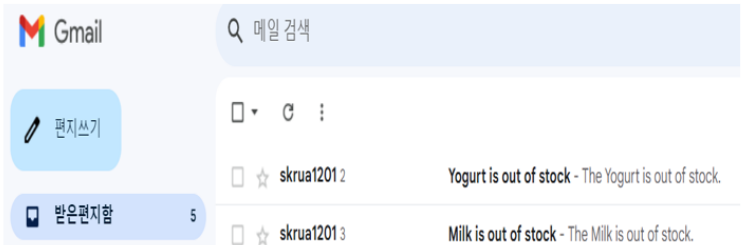
Secondly, an email is sent when an item is out of stock. As mentioned in Section 2.2.1, food may be temporarily removed and then reinserted. To handle this again, the time is set to 5 seconds, and if food is not detected within 5 seconds, an email is sent notifying that it is out of stock.



(a)



(b)



(c)

Figure 13. E-mail results

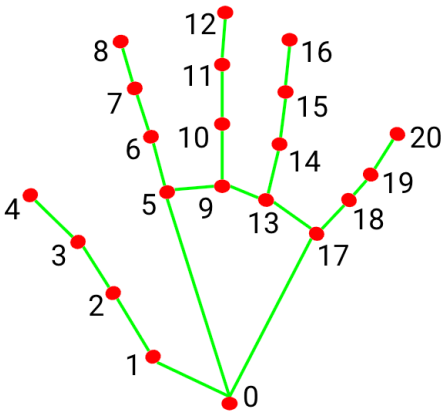
```
def send_email(subject, body):
    msg = MIMEMultipart()
    msg['From'] = email_sender
    msg['To'] = email_recipient
    msg['Subject'] = subject
```

```
    msg.attach(MIMEText(body, 'plain'))

    try:
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(smtp_user, smtp_password)
        text = msg.as_string()
        server.sendmail(email_sender, email_recipient, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

2.6. Detect the hand & Display the price

In order to check the price of food, a method was devised to find out the price by pointing to the food with the hand. This method uses mediapipe, a model that detects palms and provides 21 coordinates for each hand. The information on this is shown in Figure 12. Among these, coordinate number 8, which represents the tip of the index finger, was used. If coordinate number 8 is within the bounding box of the food, it is set to indicate the price of the food. Conversely, if it is outside the box, the price will not appear. Therefore, to know the price, you must point to the food with your hand and place the tip of your index finger inside the bounding box.

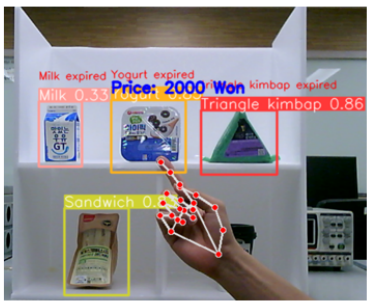


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

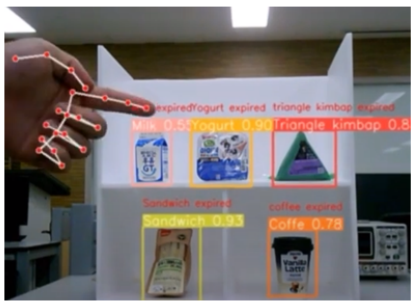
Figure 14. twenty-one coordinates on the palm of your hand

The price of triangle kimbap is 1,600 won, milk is 1,100 won, coffee is 2,100 won, yogurt is 2,000 won, and sandwiches are 2,500 won, reflecting the prices of foods sold at actual convenience stores. When you point to each food, the corresponding price is displayed on the screen. Looking at Figure 13(a) and (b), we can see that if coordinate number 8 is within the bounding box of the food, the price appears. Conversely, as shown in Figure 13(c), if the tip of the index finger is not within the bounding box of the food, the price does not appear.

(a)



(b)



(c)

Figure 15. Indicate the price of food by pointing

```
def hand_detection(frame, display_frame, results):                # Hand
detection function
    frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    hand_results = hands.process(frame_rgb)

    if hand_results.multi_hand_landmarks:
        for hand_landmarks in hand_results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(display_frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

            index_finger_tip =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
            h, w, _ = display_frame.shape
            index_finger_tip_coords = (int(index_finger_tip.x * w),
int(index_finger_tip.y * h))        # coordinates of index finger

            for result in results[0].boxes:
                class_id = int(result.cls)
                class_name = class_names[class_id]
                x1, y1, x2, y2 = map(int, result.xyxy[0])

                if x1 < index_finger_tip_coords[0] < x2 and y1 <
index_finger_tip_coords[1] < y2:       # If the finger tip is inside the contour
box
                    price = prices[class_name]
                    cv.putText(display_frame, f"Price: {price} Won", (x1, y1 -
10), cv.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2, cv.LINE_AA)
```

III. Results and Analysis

Explain the final result and analyze in terms of accuracy/precision/recall etc..

1. Result

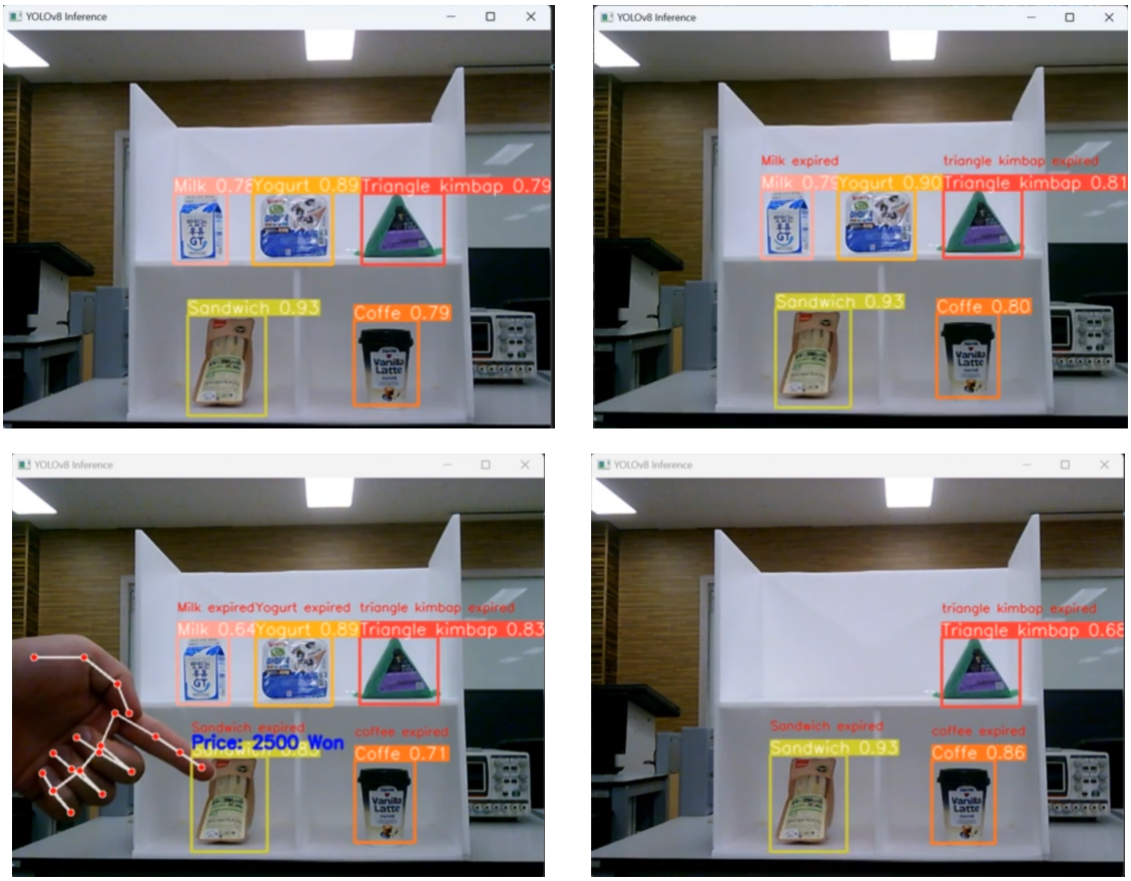


Figure 15. Result Images

The above images show item detection, expiration date check, price confirmation, and inventory management in a clockwise order. It can be seen that the algorithm in the above flowchart works well.

Demo video

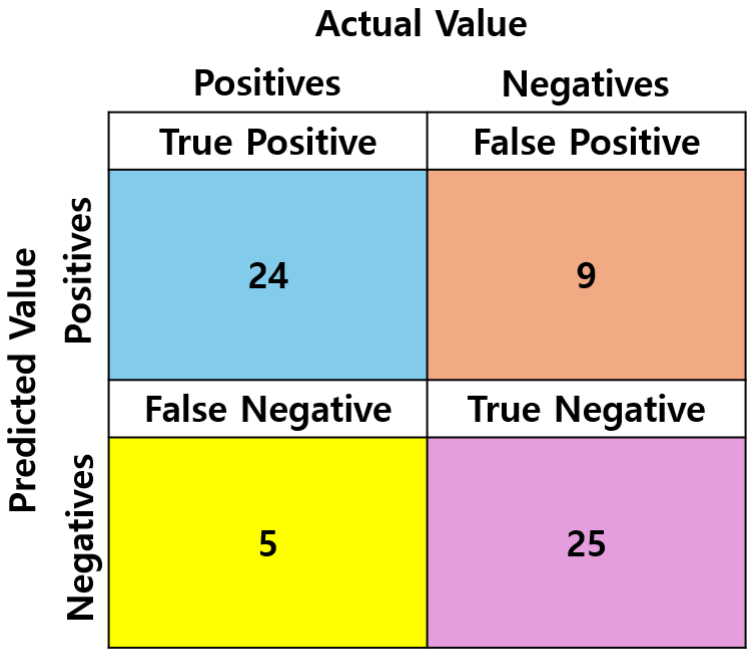
We created a demo video that shows the result of final project

Demo Video : [Youtube Link 1](), [Youtube Link 2]()

2. Evaluation

Figure 16. Images for Evaluation

The performance of the model, trained on various convenience store foods, was tested. The photos used to evaluate the model's performance consisted of both personally taken photos and those obtained from the internet.



Evaluation	Score
Accuracy	77.77%
Precision	72.72%
Recall	82.76%
F1-Score	77.41%

The model's performance evaluated with various online images was not high. The reasons for these results are as follows:

1. Lack of dataset diversity: The model trained on images taken with the same camera, in the same lighting, at a single convenience store became vulnerable as lighting and environment changed. It was not trained with datasets from various convenience stores, different lighting conditions, and different cameras.
2. Products with different designs despite being the same type: Even within the same convenience store, there are many different types of the same coffee. These products have varying shapes and colors, making it very difficult for the trained model to detect them.

Although there is a problem of not having high performance across all products in the convenience store, it is believed that higher performance can be expected if the model is trained under various conditions and on a limited range of products.

3. Trouble shooting

At first, we placed 5 foods at once and filmed the video. Through this, 3,000 to 4,000 frames were extracted and the model was trained. However, even though it was the same food appearing in the frame, it was not recognized well. So, we went to a convenience store and took pictures of various types of milk, yogurt, etc. and extracted frames. This time, rather than a mixture of different types of food in one frame, only one type of food appeared. As a result of training the

model with these new frames and adding them to the existing dataset, it was confirmed that the model recognized food well. In object recognition, it is necessary to continuously check whether more objects need additional training or not. Additionally, by separating each type of food into separate frames, the model can effectively learn the characteristics of each food.

When using DarkLabel to label and save images, I found that there was a significant difference between saving images with and without labeled bounding boxes. The bounding box was fluorescent green, and this color was included in the training model, resulting in poor object recognition. On the other hand, when the same photo was saved without bounding boxes, object recognition was much better. This means that having bounding box colors in the training data will have a negative impact on model performance. Therefore, it was confirmed that the model learns color as part of object features.

IV. Reference

- https://ykkim.gitbook.io/dlip/installation-guide/installation-guide-for-deep-learning
- https://puleugo.tistory.com/10

V. Appendix

Separate into train and verification

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * dataset split code

import os, shutil, random

# preparing the folder structure

full_data_path = 'datasets/food/Archive/milk/'
extension_allowed = '.png'
split_percentage = 90

images_path = 'datasets/food/Archive/milk/images/'
if os.path.exists(images_path):
    shutil.rmtree(images_path)
os.mkdir(images_path)

labels_path = 'datasets/food/Archive/milk/labels/'
if os.path.exists(labels_path):
    shutil.rmtree(labels_path)
os.mkdir(labels_path)

training_images_path = images_path + 'training/'
validation_images_path = images_path + 'validation/'
training_labels_path = labels_path + 'training/'
validation_labels_path = labels_path + 'validation/'

os.mkdir(training_images_path)
os.mkdir(validation_images_path)
os.mkdir(training_labels_path)
```

```
os.mkdir(validation_labels_path)

files = []

ext_len = len(extension_allowed)

for r, d, f in os.walk(full_data_path):
    for file in f:
        if file.endswith(extension_allowed):
            strip = file[0:len(file) - ext_len]
            files.append(strip)

random.shuffle(files)

size = len(files)

split = int(split_percentage * size / 100)

print("copying training data")
for i in range(split):
    strip = files[i]

    image_file = strip + extension_allowed
    src_image = full_data_path + image_file
    shutil.copy(src_image, training_images_path)

    annotation_file = strip + '.txt'
    src_label = full_data_path + annotation_file
    shutil.copy(src_label, training_labels_path)

print("copying validation data")
for i in range(split, size):
    strip = files[i]

    image_file = strip + extension_allowed
    src_image = full_data_path + image_file
    shutil.copy(src_image, validation_images_path)

    annotation_file = strip + '.txt'
    src_label = full_data_path + annotation_file
    shutil.copy(src_label, validation_labels_path)

print("finished")
```

yaml

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * yaml code for training

train:
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\coffee\images\tr
aining
```

```
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\milk\images\trai
ning
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\yogurt\images\tr
aining
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\triangle_kimbap\
images\training
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\sandwich\images\
training
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\collection\image
s\training
val:
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\coffee\images\va
lidation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\milk\images\vali
dation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\yogurt\images\va
lidation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\triangle_kimbap\
images\validation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\sandwich\images\
validation
  -
C:\Users\hilla\source\repos\DLIP_GH\Final\datasets\food\Archive\collection\image
s\validation

# number of classes
nc: 5

# class names
names: ['Triangle kimbap', 'Milk', 'Coffee', 'Yogurt', 'Sandwich']
```

Train model

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * training code

from ultralytics import YOLO

def train():
    # Load a pretrained YOLO model
    model = YOLO('yolov8s.pt')

    # Train the model using the DLIP_Final.yaml dataset for epochs
    results = model.train(data="DLIP_Final.yaml", epochs=30)
```

```
if __name__ == '__main__':
    train()
```

Main code

```
# * DLIP_Final Project_CNN Object Detection: Management of Expiration Dates and
Inventory in Unmanned Stores
# * author: Gyeonheal An, TaegeonHan
# * Date: 2024-06-24
# * Main Code

from ultralytics import YOLO
import cv2 as cv
import time
import mediapipe as mp
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# Define expiry times for each class in seconds
expiry_times = {
    'triangle kimbap': 5,
    'Milk': 10,
    'coffee': 25,
    'Yogurt': 15,
    'Sandwich': 20
}

# Define prices for each class in Won
prices = {
    'triangle kimbap': 1600,
    'Milk': 1100,
    'coffee': 2100,
    'Yogurt': 2000,
    'Sandwich': 2500
}

# Store first detection times for each class
first_detection = {}

# Store last detection times for each class
last_detection = {}

# Store email sent status for each class
expired_email_flag = {}

# Store stock empty email sent status for each class
stock_email_flag = {}

# Define class names as per the YAML file
class_names = ['triangle kimbap', 'Milk', 'coffee', 'Yogurt', 'Sandwich']

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
```

```
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=2,
min_detection_confidence=0.5)

# Email configuration (Default)
smtp_server = "smtp.naver.com"
smtp_port = 587
# Email configuration (User)
smtp_user = ""              # Naver ID
smtp_password = ""          # Naver password
email_sender = ""           # Naver E-mail address
email_recipient = ""        # Receive E-mail address

def send_email(subject, body):
    msg = MIMEMultipart()
    msg['From'] = email_sender
    msg['To'] = email_recipient
    msg['Subject'] = subject

    msg.attach(MIMEText(body, 'plain'))

    try:
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(smtp_user, smtp_password)
        text = msg.as_string()
        server.sendmail(email_sender, email_recipient, text)
        server.quit()
        print("Email sent successfully")
    except Exception as e:
        print(f"Failed to send email: {e}")

def process_frame(webcam, model):
    success, frame = webcam.read()
    if not success:
        return None, None, None

    results = model(frame)
    display_frame = results[0].plot()

    return frame, display_frame, results

def update_detection_times(results, current_time, display_frame): # Calculate
Passed Time and Determine if the expiration date has passed
    detected_classes = set()
    for result in results[0].boxes:
        class_id = int(result.cls)
        class_name = class_names[class_id]
        detected_classes.add(class_name)

        x1, y1, x2, y2 = map(int, result.xyxy[0])
        if class_name not in first_detection:                       # When it is
detected first time --> Add the class
            first_detection[class_name] = current_time              # Indexing
the first detection time for each classes
            expired_email_flag[class_name] = False                  #
Initializing the flag
            stock_email_flag[class_name] = False
        last_detection[class_name] = current_time
```

```
        passed_time = current_time - first_detection[class_name]    #
Calculating passed time

        if passed_time > expiry_times[class_name]:                  # When the
product passed the expired date
            cv.putText(display_frame, f"{class_name} expired", (x1, y1 - 30),
cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv.LINE_AA)
            if not expired_email_flag[class_name]:                  # Sending E-
mail
                remaining_times = []                                # Store the
remain times for product which not passed the expired date
                for other_class in class_names:                     # whole
classes
                    if other_class in first_detection and other_class !=
class_name:    # A class other than the currently expired product class that has
already been detected
                        remaining_time = expiry_times[other_class] -
(current_time - first_detection[other_class])  # Calculating Remain time
                        remaining_times.append(f"{other_class}:
{remaining_time:.2f} seconds remaining")
                msg_body = f"The {class_name} is expired.\n\nRemaining times for
other items:\n" + "\n".join(remaining_times)   # Message
                send_email(f"{class_name} is expired", msg_body)
                expired_email_flag[class_name] = True

    check_stock_empty(detected_classes, current_time)

def check_stock_empty(detected_classes, current_time):
    for class_name in class_names:
        if class_name not in detected_classes:
            if class_name in last_detection:
                if current_time - last_detection[class_name] > 5 and not
stock_email_flag.get(class_name, False):       # If it is not detected over
programmed times
                    send_email(f"{class_name} is out of stock", f"The
{class_name} is out of stock.")
                    stock_email_flag[class_name] = True             # E-mail
flag not to send E-mail duplicate
            else:
                last_detection[class_name] = current_time           # Update the
last detection time

    reset_detection_times(detected_classes, current_time)

def reset_detection_times(detected_classes, current_time):
    for class_name in list(first_detection.keys()):
        if class_name not in detected_classes:
            if current_time - last_detection[class_name] > 5:       # When it is
not detected over programmed times
                first_detection.pop(class_name)                     # Clear all
the product index and flags
                last_detection.pop(class_name)
                expired_email_flag.pop(class_name)
                stock_email_flag.pop(class_name, None)

def hand_detection(frame, display_frame, results):                # Hand
detection function
    frame_rgb = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
```

```
    hand_results = hands.process(frame_rgb)

    if hand_results.multi_hand_landmarks:
        for hand_landmarks in hand_results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(display_frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

            index_finger_tip =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
            h, w, _ = display_frame.shape
            index_finger_tip_coords = (int(index_finger_tip.x * w),
int(index_finger_tip.y * h))        # coordinates of index finger

            for result in results[0].boxes:
                class_id = int(result.cls)
                class_name = class_names[class_id]
                x1, y1, x2, y2 = map(int, result.xyxy[0])

                if x1 < index_finger_tip_coords[0] < x2 and y1 <
index_finger_tip_coords[1] < y2:       # If the finger tip is inside the contour
box
                    price = prices[class_name]
                    cv.putText(display_frame, f"Price: {price} Won", (x1, y1 -
10), cv.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2, cv.LINE_AA)

def main():
    model = YOLO('YOLOv8s_30.pt')

    webcam = cv.VideoCapture("test_video.mp4")
    # webcam = cv.VideoCapture(1)

    if not webcam.isOpened():
        print("Could not open webcam")
        return None

    while webcam.isOpened():
        frame, display_frame, results = process_frame(webcam, model)
        if frame is None:
            break

        current_time = time.time()                                      #
Updating Current Time
        update_detection_times(results, current_time, display_frame)    #
Expiration date management function
        hand_detection(frame, display_frame, results)

        cv.imshow("YOLOv8 Inference", display_frame)

        if cv.waitKey(3) & 0xFF == ord("q"):
            break

    webcam.release()
    cv.destroyAllWindows()

if __name__ == '__main__':
    main()
```