G-RIPS Sendai 2025

IHI Group

Final Report

Authors:
[1] Joanne Dong
[2] Benedict Ejelonu
[3] Ryo Fukano
[4] Emma Sandidge

Mentors:
[+] Masaki Ogawa
[+] Shunsuke Kano
[¤] Taka Nakayama
[¤] Ayumi Kubo
[¤] Kazuhiro Kitamura
[¤] Hidekazu Fujimura
[¤] Hideki Sakakura
[¤] Min-Hye Oh

[1] University of Michigan, Ann Arbor
[2] African Institute of Mathematical Sciences, South Africa
[3] Tohoku University, Mathematics Department
[4] Florida International University
[+] Tohoku University, Mathematical Science Center for Co-Creative Society
[¤] IHI Logistics & Machinery Corporation

August 7, 2025

Contents

1 Abstract

In this project, we design an algorithm that takes an image of a cardboard box as an input and outputs the expiration date detected on the box. This task involves identifying regions that contain dates and distinguishing them from other numerical information on the box, translating the visual information into text, and extracting dates from texts. We combine various state-of-the-art open-source softwares to create a pipeline that is easy to understand, and is well-documented for troubleshooting purposes and further development. In this report, we explain in detail how the tools we employ handle different aspects of our task and report on the overall performance of our algorithm.

2 Introduction

2.1 Problem Statement

The goal of the project is to design an algorithm that detects the expiration date from images of cardboard boxes containing food and beverage items, which more generally, can be adopted for large-scale processing perishable goods. This algorithm will eventually be implemented in-house at a distribution center. As depicted in Figure 1, a sensor will trigger a camera to take an image of the entire side of a box as it travels on a conveyor belt. This image becomes the input for our algorithm, which will search for the expiration date on the box and give the expiration date as an output to an automatic labeling machine that labels the box with the expiration date. If 600 boxes are transported along the conveyor belt per hour, this process should take at most 6 seconds per box. Our approach combines open-source softwares to achieve the goal.

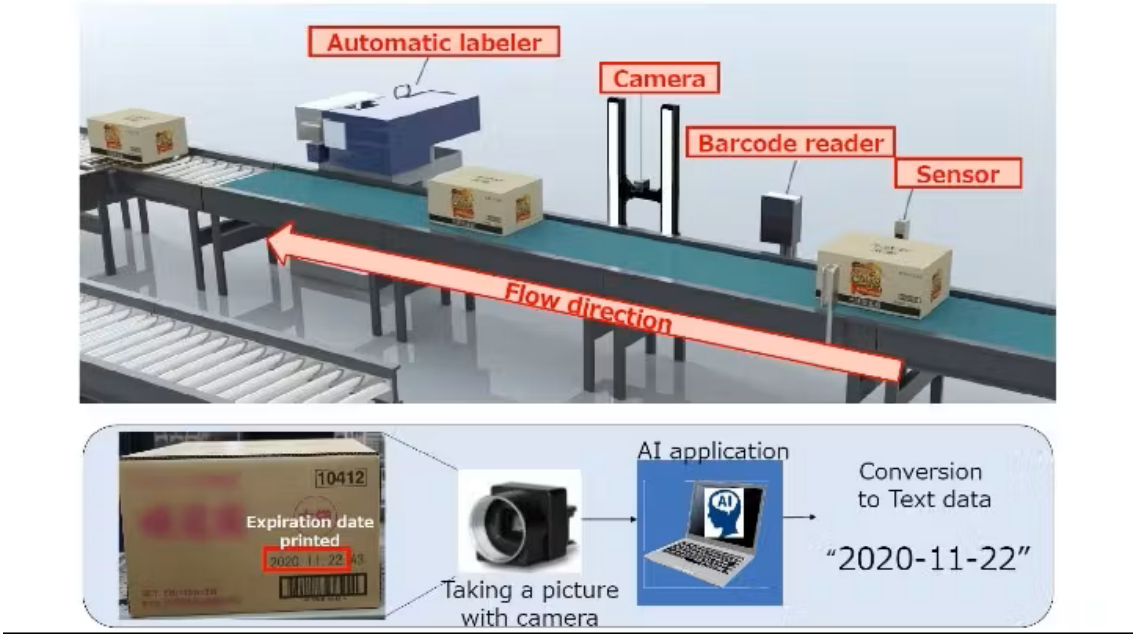


Figure 1: This figure illustrates how the automated expiry date extraction algorithm would be implemented in a distribution center. (Photo courtesy of IHI Logistics & Machinery Corporation)

2.2 Background

The task of correctly identifying an expiration date is of utmost importance for ensuring food safety for consumers and could be applied to handling other perishable goods. From a business perspective, this is beneficial because it helps the company manage a current inventory and reduce waste.

In its current form, this task is performed manually by the employees of the company. That is, an employee visually inspects the box and enters the expiration date by hand into the computer or using a handheld terminal device that can read characters if the employee aims the device where the expiration

date is located. This introduces a potential for error. Our algorithm would assist in automating this task, potentially interfacing with an entire automated warehouse system.

2.3 Preliminaries

2.3.1 Description of Data

The data set consists of 2500 images of cardboard boxes on the conveyor belt. The images contain varying lighting conditions, box orientations, and resolutions. Typically, the box has an expiration date printed on the side of the box, but the location of the date various based on the product. There are also boxes with no expiration date and boxes with multiple dates, i.e. manufacturing dates. A sample cardboard box is shown in Figure 2.

Figure 2: Sample image of cardboard box.

The expiration dates appear in different delimiter styles, such as slash(/), etc., and also without any separators at all. All dates appear in the following order: year, month, and day. Chinese-formatted dates are also considered.

2.3.2 Assumptions

To ensure that the date assignment process is uniform across all boxes, we made the following assumptions:

1. If a box contains no expiration date on the side facing the camera, it will be labeled as "None".
2. If the date contains only year and month values, we assign the day value to be the last day of the month, i.e. the date "2026.04" should be extracted as 2026-04-30.

3 Methodology

This section outlines the methodology adopted for automated expiry date extraction from carton box images. The approach combines object detection, image processing, optical character recognition (OCR), and pattern-based validation in a sequential pipeline. The methodology is designed to minimize manual intervention while ensuring robustness against diverse image qualities and layout variations.

3.1 Workflow

The contents of this section have been made non-public following consultation with the industrial mentors.

3.2 Dataset Preparation

To prepare for the object detection stage, we organized the image data into an annotated data set complete with bounding boxes and class labels. We employed Label Studio, a data labeling software [3], which allows the user to draw bounding boxes on image data and choose a corresponding class for each box. As each cardboard box contains numerous types of visual data, we decided to define three classes of information for the object detection model:

- BarCode - 0
- DateInfo - 1
- ProdInfo - 2

All barcodes on the boxes are identified with the BarCode class, any possible date information (including manufacturing dates) on the box is labeled as DateInfo, and product information such as lot number or product volume are classified as ProdInfo. An example of these class identifications is shown in Figure 3.

Figure 3: Sample image with labeled classes.

We found that including multiple classes improved the object detection stage as the model was trained both to detect possible date regions and learn what is not a possible date region. We discuss more about our different labeling strategies and how it affected the object detection stage in section 4.1. The location of each bounding box is stored in a corresponding text file in (x, y, w, h) format. Here, (x, y) is the center of the bounding box and (w, h) are the width and height, respectively. The class number is also stored for each bounding box, as seen in Figure 4.

```
1 0.3345189296207956 0.5552776191330846 0.20351175621678036 0.027518116074985385
0 0.547505929548074 0.4782751715197483 0.13018160923988376 0.05503623214997063
2 0.7839750572232466 0.5210006675309086 0.1425406349587628 0.13855612830738234
0 0.7489643720136934 0.7738711402630623 0.16602270663208935 0.06371784010757174
```

Figure 4: Bounding box coordinates for sample image.

These coordinates are used during the validation of the object detection model to see if the predicted detections are correct or not. The following section will discuss more about the object detection model and how it applies to our task.

3.3 YOLO for Object Detection

3.3.1 How it works

YOLO (You Only Look Once) [4] [5] is an object detection model based on a deep learning framework that takes an input image and outputs the coordinates of bounding boxes along with class probabilities for each detected object.

Figure 5: The structure of YOLO.

Input:

The input to the YOLO model is an RGB image represented as a 3-channel tensor of shape $H \times W \times 3$, where H and W denote the height and width of the image. Regardless of the original image format such as JPEG, PNG, WebP, or BMP, the image is decoded and converted into a standardized 3-channel RGB representation prior to inference. The image is then resized to a fixed resolution required by the model architecture. Preprocessing steps include normalization, aspect ratio-preserving padding, and, during training, data augmentation techniques such as flipping, cropping, or color jittering.

Backbone:

The backbone is a convolutional neural network (CNN) that processes the input image and extracts multi-scale local features. By applying a series of convolutional layers, the backbone transforms the raw image into a compact feature map that encodes spatial patterns such as edges, textures, and object parts.

Neck:

The neck is designed to aggregate and refine the feature maps obtained from different stages of the backbone. The module enables the model to combine low-level spatial information with high-level semantic information, which is critical for detecting objects at multiple scales and locations within the image.

Head:

The head is responsible for producing final predictions, including bounding box coordinates, objectness scores, and class probabilities.

Output:

The output consists of a coordinates of predicted bounding boxes, each associated with an object class label and a confidence score. For each detection, the model outputs the bounding box parameters in the form of center coordinates (x, y, w, h) with the probability distribution over object classes and an objectness score indicating the likelihood that an object is present in the box, where (x, y) is the center of the box, w is the width, and h is the height.

To obtain the coordinates of the four corners of the bounding box, the following conversion is applied:

- Top-left corner: $(x - \frac{w}{2},\ y - \frac{h}{2})$
- Top-right corner: $(x + \frac{w}{2},\ y - \frac{h}{2})$
- Bottom-left corner: $(x - \frac{w}{2},\ y + \frac{h}{2})$
- Bottom-right corner: $(x + \frac{w}{2},\ y + \frac{h}{2})$

Although the input image is resized to a fixed resolution, the coordinates of the predicted bounding boxes are post-processed and mapped back to the coordinate system of the original image. This ensures that detections are expressed relative to the original image dimensions, allowing for accurate localization regardless of the input resizing.

3.3.2 YOLO's metrics

Box Loss:

In YOLO models, the Box Loss quantifies the localization error between predicted bounding boxes and ground truth boxes. It is typically formulated using a distance-based regression loss. The main idea is to make the predicted box $b = (x, y, w, h)$ as close as possible to the ground truth box $\hat{b} = (\hat{x}, \hat{y}, \hat{w}, \hat{h})$.

A commonly used metric to compare two bounding boxes is the Intersection over Union (IoU). It quantifies the overlap between a predicted bounding box and its corresponding ground truth. Let B_p be the predicted bounding box and B_{gt} be the ground truth bounding box. Then, IoU is defined as the ratio of the area of their intersection to the area of their union:

$$\text{IoU}(B_p, B_{gt}) = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$

Here, $\cap$ denotes the intersection of the two boxes, $\cup$ denotes their union, and area$(\cdot)$ represents the area of a region.

IoU score ranges from 0 to 1, where IoU $= 1$ indicates perfect prediction.

While IoU is effective for evaluating overlap quality, it has drawbacks as a loss function, particularly when $B_p \cup B_{gt} = \emptyset$, leading to zero gradient. Therefore, recent YOLO models adopt IoU-based losses that improve gradient flow and better guide training.

Class Loss:

The Class Loss measures the discrepancy between the predicted class probabilities and the ground truth class labels. Unlike traditional multi-class classification, YOLO formulates the classification task as a multi-label binary classification for each anchor or detection head, using a sigmoid activation per class.

Let n be the total number of classes. For each predicted object, the model outputs a class probability vector:

$$\mathbf{p} = (p_1, p_2, \ldots, p_n), \quad p_i \in [0, 1]$$

and the corresponding one-hot encoded ground truth vector is:

$$\hat{\mathbf{p}} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n), \quad \hat{p}_i \in \{0, 1\}$$

then, Binary Cross Entropy Loss is defined as :

$$\mathcal{L}_{\text{cls}} = -\sum_{i=1}^{n} [\hat{p}_i \cdot \log(p_i) + (1 - \hat{p}_i) \cdot \log(1 - p_c)] \tag{1}$$

Finally, the total classification loss over all detected objects is averaged across the batch:

$$\mathcal{L}_{\text{cls, total}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\text{cls}}^{(i)} \tag{2}$$

where N is the number of predictions included in the batch, and $\mathcal{L}_{\text{cls}}^{(i)}$ is the Binary Cross Entropy Loss for each prediction.

Overall, the Class Loss guides the model to assign high probabilities to the correct class labels and suppress incorrect ones.

DFL (Distribution Focal Loss):

The DFL is designed to improve the precision of bounding box regression in object detection by converting the regression problem into a classification-like distribution prediction problem. It replaces direct scalar regression of box offsets with a discrete probability distribution over possible locations.

Precision and Recall (P/R):

Precision and recall are statistical measures used to quantify the quality of binary classification, adapted to object detection by treating each detected object as a classification decision. Formally, they are defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where:

- TP (True Positives): Detections correctly matching ground truth objects.
- FP (False Positives): Detections that do not correspond to any ground truth object.
- FN (False Negatives): Ground truth objects missed by the detector.

mean Average Precision (mAP) :

Mean Average Precision (mAP) is a primary metric that evaluates how accurately the model detects and classifies objects. It is computed from the precision-recall (P/R) curve, which reflects the trade-off between precision and recall at various confidence thresholds.

For a single class, let $\mathcal{P}(r)$ be the precision as a function of recall $r \in [0, 1]$. The Average Precision (AP) is defined as:

$$\text{AP} = \int_0^1 \mathcal{P}(r)\, dr$$

Given a set of classes $\{C_1, \ldots, C_n\}$, the mean Average Precision is computed as:

$$\text{mAP} = \frac{1}{k} \sum_{i=1}^{n} \text{AP}_{C_i}$$

where AP_{C_i} denotes the Average Precision for class C_i.

mAP50 and mAP50-95 :

To determine whether a predicted bounding box is a true positive, we use the IoU, Depending on the chosen IoU threshold, we define different variants of mAP:

- **mAP50** uses a fixed IoU threshold of 0.50 to determine true positives. That is,

$$\text{mAP50} = \frac{1}{k} \sum_{i=1}^{n} \text{AP}_{C_i}^{0.50}$$

  where $\text{AP}_{C_i}^{0.50}$ is the Average Precision for class C_i evaluated at IoU threshold 0.50.

- **mAP50-95** averages the mAP across multiple IoU thresholds ranging from 0.50 to 0.95 in steps of 0.05:

$$\text{mAP50} - 95 = \frac{1}{|T|} \sum_{\tau \in T} \left(\frac{1}{k} \sum_{i=1}^{n} \text{AP}_{C_i}^{\tau} \right)$$

  where, $T = \{\tau \in \mathbb{R} \mid \tau = 0.50 + 0.05 \cdot j,\ j = 0, 1, \ldots, 9\} = \{0.50, 0.55, \ldots, 0.95\}$.

  This metric provides a more rigorous evaluation by assessing both detection accuracy and localization precision across a range of IoU thresholds.

3.4 Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is a computer vision task that converts text embedded in images into machine-readable characters. As illustrated in our workflow [Figure **??**], the OCR stage is responsible for transcribing text from the enhanced regions containing expiry date information.

OCR systems typically consist of two main components: (1) *text detection*, which localizes text regions within an image, and (2) *text recognition*, which interprets the characters within those regions. Formally, given an input image I, the OCR process can be expressed as a function $f : I \to T$, where T is the resulting transcribed text string. Due to the variability in text styles, print quality, and background noise present in real-world carton images, selecting an OCR engine with strong generalization and robustness was essential for ensuring reliable performance across diverse inputs. To realize this fine balance, we evaluated two state-of-the-art open-source OCR engines, **EasyOCR** and **PaddleOCR**. Although both performed competitively, **PaddleOCR** was ultimately selected for the final pipeline due to its superior accuracy, multilingual support, and robustness in handling noisy or low-contrast regions. For completeness, we briefly describe the key features and architectures of both OCR engines below.

3.5 Preprocessing for OCR

To enhance the performance of OCR on noisy or low-contrast carton box images, we apply a sequence of image preprocessing operations to the detected regions of interest (ROIs). All preprocessing operations were implemented using OpenCV [8], an open-source computer vision library widely used for image processing tasks. These operations aim to improve text visibility, suppress background noise, and normalize input conditions for the OCR engine. The key preprocessing steps are described below:

- **Grayscale Conversion:** The first step is to reduce the color image $I_{\text{RGB}} \in \mathbb{R}^{H\times W\times 3}$ to a single-channel grayscale image $I_{\text{gray}} \in \mathbb{R}^{H\times W}$. This simplifies the image representation and reduces computational complexity while retaining essential text information. The formula used for the grayscale conversion is:

$$I_{\text{gray}} = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B$$

  where R, G, B represent the red, green, and blue channels, respectively.

- **Contrast Limited Adaptive Histogram Equalization (CLAHE):** CLAHE enhances local contrast by applying histogram equalization to small image tiles while preventing over-amplification of noise. For each tile, the local histogram H is computed and clipped at a threshold to limit contrast:

$$H_{\text{clip}}(i) = \min(H(i), T)$$

  where T is the contrast limit. The clipped histogram is then redistributed and used to remap pixel intensities within the tile, producing better visibility of faint text under uneven illumination.

- **Denoising:** To reduce noise, we apply median filtering, which replaces each pixel value with the median of its neighboring values in a sliding window (kernel) of size $k \times k$. For a pixel at location (i, j), the denoised value is given by:

$$I_{\text{denoised}}(i, j) = \text{median}\left\{I(m, n) \mid (m, n) \in \mathcal{N}_k(i, j)\right\}$$

  where $\mathcal{N}_k(i, j)$ defines the $k \times k$ neighborhood centered at (i, j).

- **Contrast, Brightness, and Sharpness Adjustment:** These global enhancements modify pixel intensity values to improve text visibility:

  – *Contrast* enhancement scales the difference between pixel values and the mean intensity:

$$I' = \alpha \cdot (I - \mu) + \mu$$

    where $\alpha > 1$ increases contrast and μ is the image mean.

  – *Brightness* adjustment shifts all pixel values by a constant offset β:

$$I' = I + \beta$$

    ensuring better visibility under dark conditions.

  – *Sharpness* enhancement emphasizes edges by applying an unsharp mask:

$$I_{\text{sharp}} = I + \lambda \cdot (I - G_\sigma * I)$$

    where $G_\sigma * I$ is the image blurred using a Gaussian kernel with standard deviation σ, and λ controls the sharpness level.

- **Morphological Operations - Erosion and Dilation:** These operations refine the shape of text components in binary or high-contrast images. Let I be a binary image and K a structuring element (e.g., a 3×3 square kernel).

  – *Erosion* removes small white noise and shrinks text contours:

$$I \ominus K = \{x \mid K_x \subseteq I\}$$

    where K_x denotes the structuring element centered at pixel x.

– *Dilation* expands text components and fills small holes:

$$I \oplus K = \{x \mid K_x \cap I \neq \emptyset\}$$

These operations are especially useful for correcting disconnected or broken character strokes before OCR.

3.6 EasyOCR

EasyOCR [7] is a modular OCR system that combines deep learning-based text detection and recognition components. Its architecture is composed of two main stages: **(1) text detection** using the CRAFT model, and **(2) text recognition** using a Convolutional Recurrent Neural Network (CRNN).

3.6.1 Text Detection - CRAFT

CRAFT (Character Region Awareness for Text Detection) predicts character-level bounding boxes and the affinity between characters to assemble full text lines. It operates by learning two heatmaps: one for individual character regions H_c, and another for character affinities H_a. These are thresholded and post-processed using connected components to produce bounding boxes.

$$H_{\text{CRAFT}} = \sigma(W * I), \quad \text{where } \sigma \text{ is the sigmoid activation}$$

3.6.2 Text Recognition - CRNN

CRNN integrates convolutional feature extraction, sequence modeling via Recurrent Neural Networks (RNNs), and transcription using Connectionist Temporal Classification (CTC):

1. **Feature Extraction:** A CNN transforms the input image $I \in \mathbb{R}^{H \times W}$ into a sequence of feature vectors $\{f_t\}$, where each $f_t \in \mathbb{R}^d$ corresponds to a vertical slice of the image.
2. **Sequence Modeling:** A Bidirectional LSTM models dependencies between time steps:

$$\mathbf{h}_t = \text{BiLSTM}(f_t)$$

3. **Transcription - CTC:** The network outputs a probability distribution over characters for each time step. The final text prediction y is decoded by maximizing the probability under CTC loss:

$$\mathcal{L}_{\text{CTC}} = -\log p(y|x)$$

where x is the feature sequence and y the target label sequence.

3.6.3 EasyOCR - Performance Summary

EasyOCR is quick to setup, requires minimal tuning, and performs well on clean or moderately noisy inputs. However, it showed reduced reliability in our use case when dealing with:

- Low-contrast or blurred expiry date regions.
- Complex background textures or overlapping printed text.
- Images having dot matrix characters.

3.7 PaddleOCR

PaddleOCR [6] is an end-to-end OCR system built on the PaddlePaddle deep learning framework. It offers modular and highly configurable components for **text detection**, **text recognition**, and **post-processing**, enabling high-performance transcription across multilingual and visually noisy datasets. This modularity made it a particularly strong candidate for our pipeline. Below, we describe the specific components employed in our workflow.

3.7.1 Text Detection - DBNet (Differentiable Binarization)

PaddleOCR uses DBNet for text detection, which reframes the task as a segmentation problem. The network predicts a probability map $P(x)$ and a threshold map $T(x)$ over image pixels. The final binarization output is computed using:

$$B(x) = \frac{1}{1 + \exp(-k(P(x) - T(x)))}$$

where k is a hyperparameter that controls the sharpness of the binarization. This differentiable formulation allows the model to learn sharper boundaries between text and background, which is critical for accurate detection under noise, glare, or occlusion.

3.7.2 Text Recognition - SRN and CRNN

For text recognition, PaddleOCR supports both traditional and advanced backbones:

- **CRNN:** Combines convolutional feature extraction with BiLSTM and CTC decoding, and serves as the default lightweight option.
- **SRN (Sequence Recognition Network):** A more sophisticated model that includes:
  1. **Visual Feature Extraction** via CNN.
  2. **Global Semantic Modeling** using Transformer-based self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  3. **Parallel Visual Decoding** for efficient sequence generation.

  SRN's ability to model context across a sequence enhances its robustness to partial occlusion and distorted text patterns, which were present in many of our samples.

Following character prediction, PaddleOCR applies language-aware post-processing. This includes lexicon matching, rule-based heuristics, and a lightweight character-level language model. These steps are particularly useful for filtering improbable strings and correcting minor recognition errors which turned out to be a necessity in expiry date recognition where numeric integrity is essential.

PaddleOCR was ultimately selected for its:

- Robust handling of low-contrast, noisy, and multilingual text.
- Flexible model architecture with interchangeable detection and recognition modules.
- Superior recognition accuracy on carton-based expiry dates compared to EasyOCR.

The comparison between EasyOCR and PaddleOCR's performance is presented in section 4.2.

3.8 Post-processing

After the OCR engine detects the text within each ROI, it outputs a list of strings for each line of text that it reads. This becomes the input for our post-processing stage, which begins by seeking out potential dates in the strings by searching for patterns matching the variable date formats. This job is handled by regular expressions using the `regex` package in Python. We can capture the variable date formats using two patterns:

- Pattern 1 deals with date formats that uses separators. It matches sequences with the following order reading from left to right:
  1. 2 or 4 digits representing the year, prioritizing 4 digits if possible
  2. a delimiter

3. 1 or 2 digits representing the month
4. (optional) a delimiter
5. (optional) 1 or 2 digits repesenting the day

Since the last two groups are optional, this pattern can detect both date formats including days and also those without days. We expand the delimiters in our pattern from those that are typical in date formats to account for possible misreadings from OCR text detection (e.g. a period may be misread as a comma). For the same reason, the two delimiters in this pattern don't necessarily need to be the same.

- Pattern 2 deals with date formats *without* separators. It naively looks for numerical sequences with a length of 5 to 8 digits. An experienced reader may notice that pattern 1 can easily be modified to include such sequences. However, this should genuinely be handled separately because given a sequence of a specified length, there could be more than one way to read a valid date. For instance, "21122" can be read as 2021-1-22 or 2021-12-2. With this motivation, we consider all possible date formats that could be associated to a sequence without separators.

As we tested the pipeline, we noticed that OCR would often misidentify the Chinese year character 年 (nen) as a 4, leading to incorrect dates. Anticipating this issue, we introduce a third pattern:

- Pattern 3 adds flexibility in reading dates using Chinese character separators by accounting for the possibility of misreading 年 as a 4, using the Chinese month character 月 (tsuki) as an indicator that this is a date in that format. It matches sequences with the following order reading from left to right:

1. 2 or 4 digits representing the year, prioritizing 4 digits if possible
2. 年 or 4
3. 1 or 2 digits representing the month
4. 月
5. (optional) 1 or 2 digits representing the day

We apply these patterns to find all potential dates for each ROI and compile all matches from all ROIs into a single list for comparison. Now we must consider that not all sequences of text that match the patterns are actually dates. Sometimes, product lot numbers may be picked up in the ROI. Moreover, according to our labeling convention described in section 4.1.1, 6-digit sequences, such as product numbers, are also labeled with the class DateInfo. To filter out sequences that are not dates, we use the `datetime` package, which supplies classes for manipulating dates by casting them as date objects. We also put a cap on the latest year to consider for expiration dates.

If none of the sequences matching the regex patterns are legitimate dates, the algorithm stops and outputs "None". On the other hand, multiple dates can be detected. Often, this occurs when the manufacturing date is also printed on the box. Conveniently, date objects in the `datetime` package can be ordered, which provides an elegant way to find the latest date out of all dates read from the box image. We format the latest date in a manner that is consistent across all boxes and this becomes our final output from the algorithm. We chose to format the dates as YYYY-MM-DD, but this can easily be changed by the user.

3.9 Definition of Accuracy

The main metric of interest in evaluating the overall performance of our algorithm is the accuracy, which measures whether or not we extract the expiration date correctly when the date is visible in the image of the box. Sometimes, the wrong side of the box faces the camera and the expiration date is not visible, and in this case it would be correctly be labeled as "None" so that it is sent to manual inspection. All possibilities and whether or not the case is counted as a correct extraction are summarized in Figure 6. If N_{boxes} is the total number of boxes whose images have been processed, the accuracy of the algorithm is the number of correct dates extracted according to Figure 6 divided by N_{boxes}.

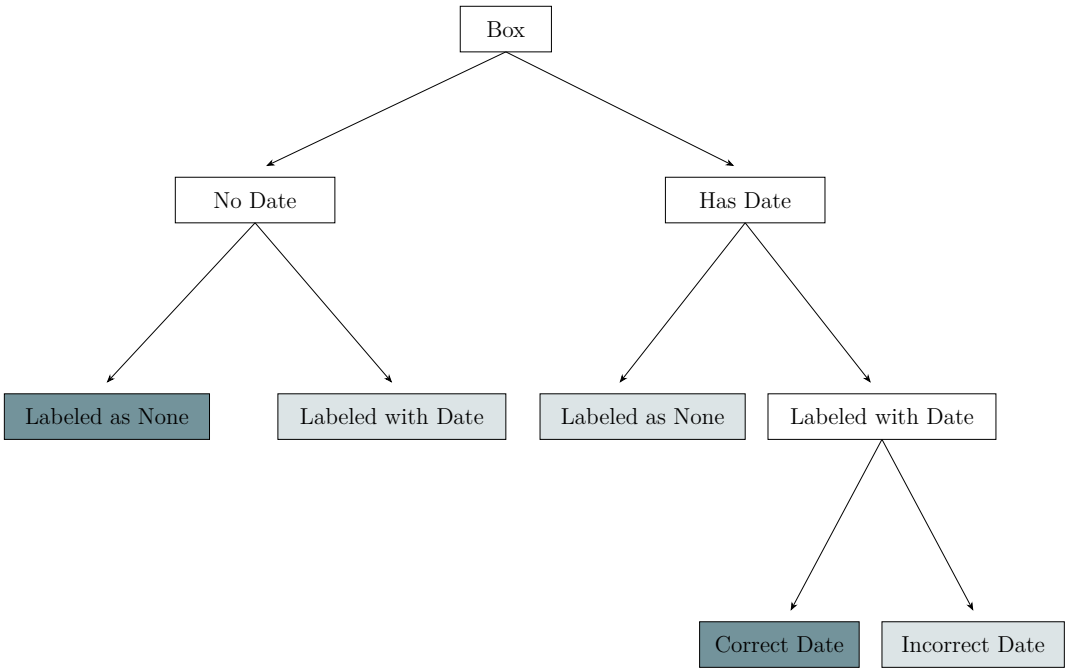


Figure 6: Pictured above is a decision tree that illustrates all possible scenarios encountered when determining the expiration date from a box. The cases with a darker background are counted as correct while cases with a lighter background are considered incorrect.

Having defined accuracy, we now define what it means for an extracted date to be *correct*. For this, we consider two notions of correctness:

1. **YMD accuracy**: correctly identifying the year, month, and day, adhering to our assumption that the day must be the last day of the month when it is not specified
2. **YM accuracy**: correctly identifying only the year and month

This distinction is interesting to consider from both technical and business perspectives. As we were developing the algorithm, we noticed that the most challenging part of the dates to detect were the days. One example of how this may arise is that when no day is specified, other numbers following the date could be picked up as the day. (Explicit examples are provided in section 4.3.1.) As we tested our algorithm, we wanted to compare the YMD and YM accuracies, pinpoint patterns in cases where the day was misdetected for troubleshooting, and try to bridge the gap between the two types of accuracies.

From a business standpoint, it is valuable to know both the YMD and YM accuracies in light of the 1/3 rule in the Japanese food industry, which divides the time between manufacturing and expiration of a product into thirds for manufacturers, retailers, and consumers. For example, if a product expires within 6 months, the manufacturer must ship out the product within 2 months after production to retailers. Then retailers have 2 months to bring the product to consumers (sell-by date). The product will expire 2 months after the sales deadline. So while YM accuracy may suffice at the manufacturer to retailers stage, YMD accuracy is crucial for consumers.

4 Results

This section presents the results obtained from our pipeline, including detection and recognition performance under varying configurations. All experiments and training routines were executed on an NVIDIA A100 GPU with 40GB of memory. Input images had an average resolution of 2448×2048 pixels (height $\times$ width). For the YOLO module, two training setups were explored: (i) fine-tuning a pretrained YOLO model for 100 epochs, and (ii) training a YOLO model from scratch for 200 epochs. After training and evaluation,

the complete pipeline was deployed as a command-line interface (CLI) application to support scalable and reproducible inference.

4.1 YOLO Results

4.1.1 Labeling Strategies

We experimented with several different labeling techniques to see what made the YOLO model perform best. First, in addition to labeling dates on the boxes as DateInfo, we also labeled other strings of numbers that had styles similar to the date formats. An example is shown in Figure 7.

Figure 7: Box image with "205410" labeled as DateInfo.

We found that it was more effective to teach the model to recognize strings of numbers that could be possible dates than strictly just the expiration dates. As mentioned in section 3.8, we can filter out strings of numbers that are not valid dates, so in this case the detection "205410" would be removed as "54" is not a valid month. In addition, some boxes contain date information that is two lines. Our first approach was to label the information as DateInfo with one bounding box. However, when the ROI was transferred to the next stage of the pipeline, it was difficult to extract text from the cropped region. So, we decided to create a "line-separated data set", which means we would use two separate bounding boxes to label date information that took up two lines. Figure 8 provides an example image with two lines of date information and how we changed our labeling process.

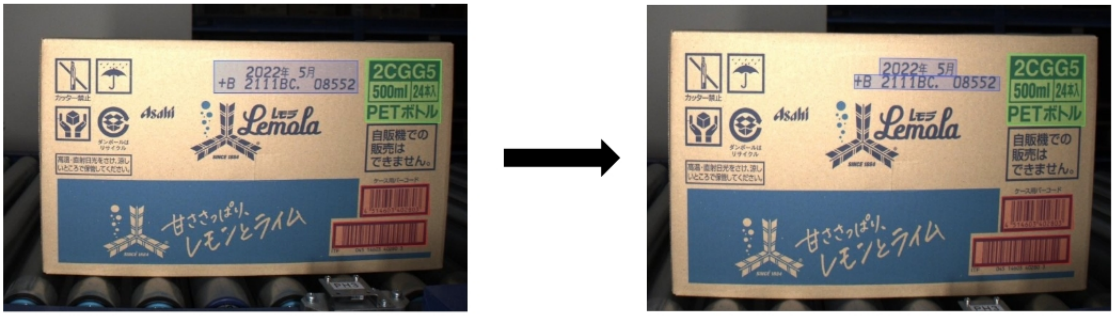
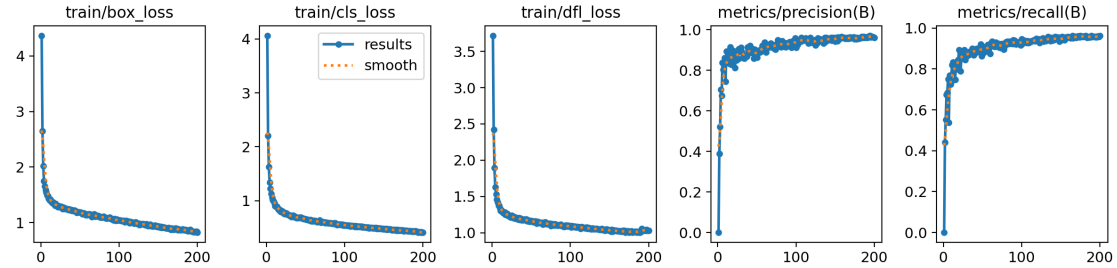


Figure 8: Example of line-separated labeling.

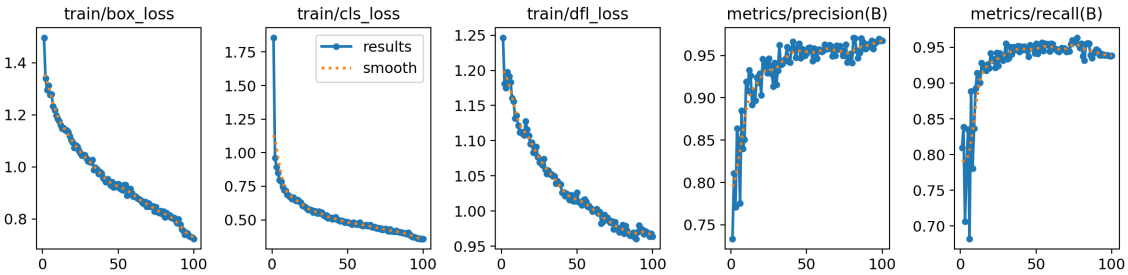
This labeling strategy trained the YOLO model to detect single lines of text, which means a more precise cropped image was passed to the next stage of the pipeline. The following results of the YOLO models are from training on this line-separated data set.

4.1.2 Model Performance

We trained both YOLOv8 [4] and YOLOv11 [5], where v8 and v11 represent version 8 and version 11, respectively. Among these two models, we also experimented with the size of the model. We tested YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), and YOLOv11m (medium). Typically, there is a trade-off between model size and accuracy performance. This means that using a larger model can yield high accuracy, but requires longer training times. A smaller model can train more efficiently, but it may have a worse accuracy performance. We used YOLOv8 because it is well known to have a good balance between speed and accuracy performance, and we used YOLOv11 because it is the latest version in the series of YOLO models. Our training approaches included training from scratch and transfer learning. Training from scratch means that the model is learning everything about the task from the beginning. Technically, this means that almost all the model parameters are variables, and they must be optimized to our specific goal. Transfer learning involves using a pretrained model that has been trained on a large general data set. Many of the YOLO variants have pretrained models that can be applied to different tasks. In transfer learning, most of the model parameters are fixed, and only some are changed to fit our specific data set. First, we compared the training from scratch and transfer learning methods using the YOLOv8 model. In Figure 9, we compare the training results using a YOLOv8m model.



(a) YOLOv8m with training from scratch.



(b) YOLOv8m with transfer learning.

Figure 9: Comparison of training results with training from scratch and transfer learning methods.

In the first three graphs from Figures 10(a) and 10(b), the goal is to minimize the different losses that YOLO uses for training. As defined in section 3.3.2, we have box loss, class loss, DFL, precision, and recall. As the model minimizes these losses, its predictions of bounding boxes and class improve. In our case, DFL is useful because some of the boxes have no expiration date, but still contain barcodes and product information. So, there is a class imbalance between DateInfo, BarCode, and ProdInfo. This loss helps by adding weight to those more challenging instances, which improves detections. In the remaining two graphs from Figures 10(a) and 10(b), the goal is to maximize the different metrics YOLO uses to measure model performance. When comparing the performance of the two training methods, we see that training from scratch needed 200 epochs to minimize the losses and maximize the metrics. Since there were more variable parameters, it took longer for the model to become proficient in the object detection task. When we used transfer learning, only 100 epochs were necessary to achieve good results. Using a model that has been pretrained on general data was well suited for our goals. From this point onward, we used only pretrained models, i.e. the transfer learning method. The next results shown in Figure 10 are from the YOLOv11m model using an image size of 1280×1280.

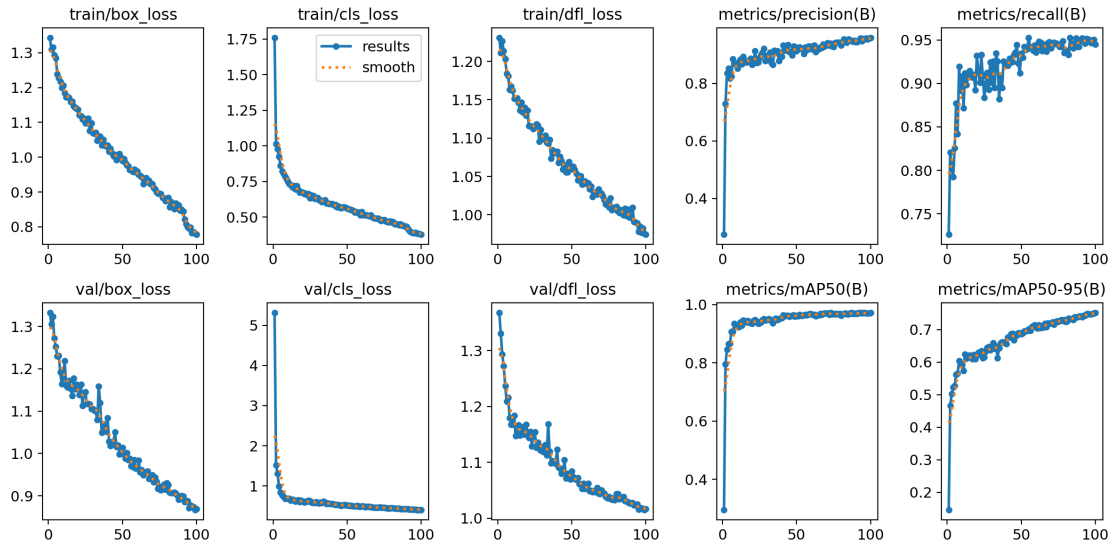


Figure 10: Results from YOLOv11m with transfer learning.

Again, over 100 epochs we were able to minimize the three losses and maximize precision and recall during training. The second row of graphs contains the results during the validation phase. Validation is used as checkpoints for the model to see how it performed on data that it has not seen before. So, minimizing these losses means that the model is performing well. We also report the mAP50 and mAP50-95 performance metrics, as defined in section 3.3.2. We see that mAP50-95 reaches about 0.79 over 100 epochs, indicating strong performance in detection and localization. Next, we share the normalized confusion matrices from YOLOv8 and YOLOv11 models in Figure 11.

(a) YOLOv8m, transfer learning.

(b) YOLOv11m, transfer learning.

Figure 11: Comparison of predictions from v8 and v11 models.

Here we use the same class IDs as mentioned in section 3.2. There is also BG for background data on the image. The rows of the matrix represent the predicted classes of the data, and the columns are their true values. So, the goal of a confusion matrix is to be as diagonal as possible so that the predictions match the true values. For both models, all three classes were correctly predicted above 90% of the time. For DateInfo, we predicted the correct class 97% of the time. Even though there were mispredictions between ProdInfo

and background, this does not pose an issue since the next stage concerns the DateInfo ROIs only. Using three classes was beneficial as it taught the model how to recognize text that was a possible expiration date verses product numbers and lot numbers. The following section moves into the next stage of the pipeline, where we extract text information from the YOLO ROIs.

4.2 OCR Results

This section contains results obtained from experimenting with EasyOCR and PaddleOCR models.

4.2.1 EasyOCR

Initially, we applied EasyOCR to extract characters from the ROIs. During the experiment, we observed several recognition errors, as detailed below. To ensure that only the essential information is extracted, we constrained the OCR using an allowlist: `allowlist='0123456789 年月日 +./-)(:'`.

Poor Quality of ROIs

In some packaging designs, the expiration date is printed in a tiny font. Consequently, the extracted ROI becomes a small region, often with low resolution. EasyOCR performs poorly on such low-resolution inputs:

Ground Truth:	2024.8.1/G
Prediction:	24481/

Figure 12: Low Resolution

To address this issue, we devised a preprocessing pipeline that combines grayscale conversion, denoising, sharpening, CLAHE, and contrast adjustment. The result of applying this preprocessing method to the same image is shown in the figure below:

Ground Truth:	2024.8.1/G
Prediction:	2024 8 1/6

Figure 13: Preprocessed Low Resolution

Dot Matrix

Dot matrix printing, which stands for characters using sparse dot patterns, often leading to misrecognition of characters for EasyOCR.

Ground Truth:	2025 年 04 月 +S-2405 D048091B
Prediction:	)1/) . ::2:: 7- . 年 (.

Figure 14: Dot Matrix

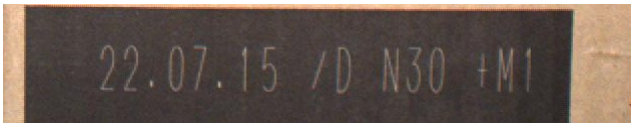
To treat this issue, we devised a preprocessing method to improve the recognition of such characters. This method employs a morphological operation called closing to bridge the blank spaces between dots. The following shows the preprocessed image, and the EasyOCR results:

Ground Truth:	2025 年 04 月 +S-2405 D048091B
Prediction:	2025 年 04 月 64398

Figure 15: Preprocessed Dot Matrix

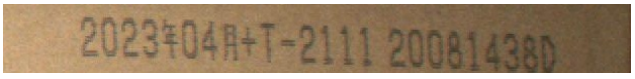
Frequently Misrecognized Characters

we also observed the following typical mispredictions of EasyOCR.



Ground Truth:	22.07.15 /D N30 +M1
Prediction:	22.01.15/010 +

Figure 16: 7 vs. 1



Ground Truth:	2023 年 04 月 +T-211120081438D
Prediction:	2023404 月-1-2111200814380

Figure 17: 年 vs. 4

Ground Truth:	2024.5.14
Prediction:	2024.5.11

Figure 18: 4 vs. 1

4.2.2 PaddleOCR

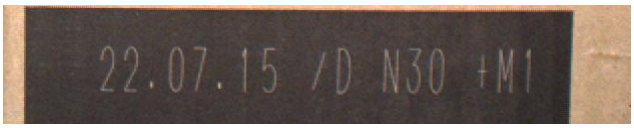
PaddleOCR performed very well and proved effective in overcoming the mispredictions encountered when using EasyOCR. Below are the results of applying PaddleOCR to the same set of images presented in the previous section. Here, all inputs to PaddleOCR are raw (i.e., without any preprocessing), and no character allowlist constraints are applied.

Ground Truth:	2024.8.1/G
Prediction:	2024.8.1/G

Figure 19: PaddleOCR for Low Resolution

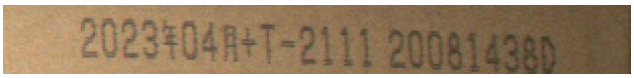
Ground Truth:	2025 年 04 月 +S-2405 D048091B
Prediction:	2025 年 04 月 +5-2405 D048091B

Figure 20: PaddleOCR for Dot Matrix



Ground Truth:	22.07.15 /D N30 +M1
Prediction:	22.07.15 /D N30 +M1

Figure 21: PaddleOCR for 7 vs. 1



Ground Truth:	2023 年 04 月 +T-211120081438D
Prediction:	2023 年 04 月 +T-211120081438D

Figure 22: PaddleOCR for 年 vs. 4

Ground Truth:	2024.5.14
Prediction:	2024.5.14

Figure 23: PaddleOCR for 4 vs. 1

As the examples show, the OCR output does not necessarily match the desired final format. In many cases, the output includes numeric sequences unrelated to date information, and the formatting of actual dates varies depending on the design of each cardboard box. Therefore, in order to extract the target date information and standardize its format from such inconsistent OCR results, we apply a post-processing method, which is described in the following section.

4.3 Regex Results

4.3.1 Examples

To illustrate how the regex patterns match detected text from OCR, we look at a few examples of text detections from OCR and show how we use the regex patterns to pick out potential dates.

Example	OCR Text Detection	regex Matches	Extracted Date(s)
1	2022.04.13.)78.1554	2022 . 04 . 13 78 . 15 54	2024-04-13
2	88/20220515/0055	88 / 20 22 0515 / 00 55 20220515	2022-05-15
3	G/2022.11.28/2021.12.0414562	2022 . 11 . 28 2021 . 12 . 04 0414562	2022-11-28 2021-12-04

Table 1: Examples of what regex patterns introduced in section 3.8 pick out from real OCR outputs. The shading in the "regex Matches" column distinguishes the groupings in the patterns.

As the text is parsed from left to right, any characters that are matched are "consumed", meaning that there cannot be overlapping matches for a single pattern. In example 1, we see that there are two patterns that match pattern 1. After both candidates are converted into datetime objects, we see that the second match, with year = 78, month = 15, and day = 54 is not a valid date.

In example 2, pattern 1 picks out 2 sequences that are not valid dates and these matches consume all of the characters. However, because we have a separate pattern that matches sequences without separators, pattern 2 picks up the sequence we are interested in and correctly identifies that date as 2022-05-15.

In example 3, pattern 1 picks up the 2 dates of interest, but pattern 2 also picks up the sequence following the second separator. However, this is not an issue, because it cannot represent a valid date, so this will be filtered out. Since this example returns two valid dates, the next step in post-processing will select the latest one as the expiration date.

4.3.2 Challenges

Greediness. Our regex patterns are greedy, meaning it will match as many characters as possible in the pattern for each group. As mentioned earlier when introducing the patterns in section 3.8, pattern 1 will try to match 4 digits when possible over 2 digits for the year. Similarly for the month and day, regex will prioritize matching 2 digits over 1 digit when possible. Here are some examples of edge cases when this philosophy poses an issue.

Example: "2024816" detected as "2024-08-16" when the true date is "2024-08-01" and "6" is part of a product or lot number following the date. Without context, such additional separators, distinguishing the date from the product or lot number, there is no way for us to tell why the 6 at the end should or shouldn't be included in the day by only considering the text output from OCR. The same issue arises when the expiration date is followed by the manufacturing date, and both dates do not have separators; pattern 1 consumes the first 8 digits as the first match, and continues to consume as many possible for the next match.

Example: "220608408" returns "None" when the true date is 2022-06-08. This sequence would be picked up by pattern 2, which matches as many characters as possible and extracts the sequence of maximal length, which is 8 digits. The only date format corresponding to a sequence with 8 digits is YYYY-MM-DD. The combination of year=2206, month=08, day = 40 fails to be converted into a date object because it does not represent a valid date, so the algorithm returns "None".

Missed separators. When the separator is too faint for OCR to pick it up, the date could possibly be misread. Pattern 1 allows for the second separator to be optional, allowing for dates that only specify year and month. However, the first separator between the year and month is required, so when it is missing, the first four digits are considered for the year.

Example: "2204.18" returns "None" when the true date is 2020-04-1. This occurs because the pattern picks up year=2204 and month=18 as the candidate date, but this is not a valid date.

False separators Sometimes the expiration date and manufacturing appear next to each other and are separator by one of our common separators, such as "/". Because pattern 1 allows for possible mismatch of separators, it may pick up the digits follow the second separator as the day when no actual day is specified.

Example: "2022.12/2021.11" detected as "2022-12-20" when the true expiration date is "2022-12-31", and the manufacturing date is 2021-11-30.

Other 6-digit numbers. Recall our labeling convention of selecting sequences with 6 digits as DateInfo even when they represent product numbers. A consequence of this action is that in some rare cases, when no actual date is present on the side of the box that has been photographed, the 6-digit product number could actually represent a valid date. Since there is no other bounding box labeled as DateInfo, this product number is returned as the expiration date.

4.4 Overall Performance

Now, we discuss the overall performance of our pipeline in extracting the correct expiration date from a given input image in the data set. As mentioned in section 3.9, we provide two accuracy metrics for the pipeline: **YMD Accuracy** and **YM Accuracy**. In the testing stage, we took a subset of 1000 randomly chosen images from our data set and ran the full pipeline on them. The model renames each image with the extracted expiration date, and we used this to manually validate the number of correctly identified expiration dates. We used this testing process on our pipeline with each OCR engine, EasyOCR vs. PaddleOCR. Table 2 shows a comparison of the overall performance on the subset of test images.

OCR Engine	YMD Accuracy (%)	YM Accuracy (%)
EasyOCR	70.8	76.2
PaddleOCR	96.7	98.1

Table 2: Comparison of overall accuracy of the pipeline using different OCR engines.

Here, we can see the clear increase in model performance when the PaddleOCR engine was used. Many of the challenges from OCR and regex described in sections 4.2 and 4.3 were resolved when using the PaddleOCR model. For the PaddleOCR model, the execution time was 198.27 seconds for a batch of 1000 images. For single image processing, if we are able to cache the weights for the PaddleOCR model, the processing time is 4.2 seconds per image. If we initialize the model each time an image is processed, it takes 11 seconds per image. The final prototype we have developed uses the YOLOv11m model trained on image size 1280 ×

1280 and the PaddleOCR engine. We also developed a method for visualizing images in the data set with their extracted expiration date and OCR confidence level displayed on them. In Figure 24 we share a sample output of the model using this visualization method.

Figure 24: Visualized output with extracted date and confidence score.

We see that the model can confidently read the expiration date and adds the last day of the month for the day value. In addition, the model also succeeded on edge cases such as upside down boxes, high contrast images, boxes with two dates, dates with small fonts, and dates with white text on a black background. By the request of the industrial mentors, these visualized outputs are omitted from this report. In general, the pipeline can accurately and efficiently process images to extract their expiration dates. The next section discusses future directions and ideas for this application.

5 Future Work

Another aspect that a distribution center may be interested in extracting from the images of the cartons is the barcodes, which uniquely identify the products. For distribution centers, the barcodes of interest would be the ITF-14 barcode, which carries the Global Trade Item Number (GTIN)[9]. At the logistics level, the GTIN-14 identifier is carried by the ITF-14 barcode and contains:

- the packaging indicator: describes the packaging level (inner or outer carton, pallet, etc.)
- GS1 (Global Standard 1) company prefix: uniquely identifies the company
- item reference number
- packaging indicator

Our pipeline can easily be adapted for this task, because in the first stage using YOLO detection, we already created a class for barcodes. The next stage would filter for the BarCode class instead and send those cropped detections to OCR for text detection. Post-processing would be adjusted to check for acceptable formats according to the standards.

For training and testing, we worked on a NVIDIA A100 Tensor Core 40GB GPU owned by Tohoku University and this was very efficient. However, these are expensive and it would be worthwhile to investigate the lowest configuration necessary to run our existing algorithm, including running on a CPU. One should consider the cost and performance trade-off that is suitable for the distribution center's needs.

6 Conclusion

For the purpose of ensuring food safety, we developed an algorithm that accurately extracts expiration dates from images of cardboard boxes. This date will be automatically labeled onto the box soon after the date is detected as the box travels along a conveyor belt in the distribution center. This is only one component of an automated warehouse system. While we were able to achieve high accuracy with training from images taken in a controlled environment, we also make suggestions for how to expand this setup to work in other distribution centers and for effective deployment for real-time extraction.

References

[1] *IHI, Group Company, and U.S. Startup Jointly Develop World' s First Automatic Expiration Date Reading System for Food Shipping Cases - New system leverages AI to recognize print patterns and boost productivity, reduce manpower, and cut food waste* -. https://www.ihi.co.jp/en/all_news/2021/industrial_general_machine/1197369_3364.html. Apr. 2021.

[2] *Neurala and IHI Logistics Machinery Partner to Deliver First-of-its-Kind AI in Materials and Logistics Handling*. https://www.neurala.com/press-releases/neurala-and-ihi-logistics-machinery-partner-to-deliver-first-of-its-kind-ai-in-materials-and-logistics-handling. Apr. 2021.

[3] Maxim Tkachenko et al. *Label Studio: Data labeling software*. 2020-2022. URL: https://github.com/heartexlabs/label-studio.

[4] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[5] Glenn Jocher and Jing Qiu. *Ultralytics YOLO11*. Version 11.0.0. 2024. URL: https://github.com/ultralytics/ultralytics.

[6] Cheng Cui et al. *PaddleOCR 3.0 Technical Report*. 2025. arXiv: 2507.05595 [cs.CV]. URL: https://arxiv.org/abs/2507.05595.

[7] JaidedAI. *EasyOCR*. https://github.com/JaidedAI/EasyOCR. GitHub Repository, version 1.7.2, accessed 2025. 2025.

[8] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[9] *ITF-14 Barcodes*. URL: https://www.gtin.info/itf-14-barcodes-gtin-info/. (accessed: 2025-07-30).