Human-Assisted Labeling System

BY

Chavadon Nuntchaipruck 60011237

Adarsh Tiwari 60011206

Nabin Sharma 60011231

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF ENGINEERING IN KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2020

SCHOOL OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

PROJECT CERTIFICATE

Project Title	Human-Assisted Labeling System
Student Name	Mr. Chavadon Nuntchaipruck Student ID. 60011237 Mr. Adarsh Tiwari Student ID. 60011206 Mr. Nabin Sharma Student ID. 60011231
Degree	Bachelor of Engineering in Computer Innovation Engineering

Project Advisor Signed: Orathai Sangpetch

Asst. Prof. Dr. Orathai Sangpetch

Project Advisor Signed:

Asst. Prof. Dr. Akkarit Sangpetch

Project Title	Human-Assisted Labeling System
Student Name	Mr. Chavadon Nuntchaipruck Mr. Adarsh Tiwari Mr. Nabin Sharma
Degree	Bachelor of Engineering in Computer Innovation Engineering
Project Advisor	Asst. Prof. Dr. Orathai Sangpetch International Program, Ph.D.
Project Advisor	Asst. Prof. Dr. Akkarit Sangpetch International Program, Ph.D.
Academic Years	2020

ABSTRACT

Traditionally, the process of creating an accurate labelled dataset requires multiple steps. Additionally, when multiple labelers are involved, there will be multiple inconsistencies leading to mistakes. The objective of our Human-Assisted Labelling system is to provide an application that allows user to automate the image labelling process and store labelled information to track the inconsistencies between labelers. Our system also provides a convenient interface for clustering the dataset as well as aiding labelers in the labelling process. Our system helps to automate many preparatory steps such as extracting features, removing noisy datapoints, and maintaining validation in the dataset (showing tagging inconsistencies). Additionally, noisy, or bad images from the dataset will be removed with 99% precision rate.

ACKNOWLEDGEMENTS

Firstly, we would like to express our gratefulness to our advisors Asst. Prof. Dr. Akkarit Sangpetch and Asst. Prof. Dr. Orathai Sangpetch for their support and feedback throughout the journey of this research. Their immense knowledge and experience helped us identify the problems and decipher them strategically. Secondly, we would like to thank our Research Assistant Mr. Parmer Daengphruan for his comment and feedback on our research. In addition, we would like to be grateful to our CIE friends for motivating us to complete our research work.

Lastly, we would like to thank all of our committee members for initiating the idea of the Capstone Project.

Mr. Chavadon Nuntchaipruck

Mr. Adarsh Tiwari

Mr. Nabin Sharma

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
CIE	Computer Innovation Engineering
SIIE	School of International Interdisciplinary Engineering Programs
Thaibev	Thai Beverage Public Company Limited
ResNet	Deep Residual Network
DNN	Deep Neural Network

CHAPTER 1
INTRODUCTION

This chapter introduces the overarching themes of this report and places the motivation for the work into context. Thereafter, the rationale and goals defined for the investigation of the project are discussed, followed by a summary of the overall project. Finally, an overview of the dissertation is given on a per-chapter basis.

1.1 Background

Thai Beverage or in short ThaiBev, is one of the largest beverage companies in Southeast Asia. ThaiBev is continuously trying to improve its time consumption in manufacturing and recycling bottles. One of the essential processes includes the bottle screening process which requires extensive manpower because there lies a great responsibility and decision for categorizing condition of bottles. Categorizing bottles into different classes is a crucial and essential task because lack of proper decision may end up wasting ThaiBev useful resources such as good condition bottle and time.

For instance, if a bottle has fair condition, which is needed only cleaning process before recycling, but it is categorized as bad condition bottle, then the bottle must be broken down into pieces, melted down, along with removing impurities and blown into the desired shape. In fact, refillable bottles consume 93% less energy and 47-82% less water than making a new bottle [1].

In addition, this is a manual and tedious task and humans are different towards the decision-making process because everyone has a different perspective about which bottle's condition can be categorized as dirty, clean, minor scratch, deep scratch, dusty etc. Also, to make accurate decision by human, bottle is required to check in every view such as top, bottom and side view to ensure its condition, which is dramatically decrease productivity in recycling bottle in Thaibev, so making the automate system that precisely and faster detection for categorized bottle condition, which will make Thaibev generate better performance and reduce wasting their valuable resources on recycling process.

Moreover, each view can contain all the tags. And some of the Category Group are shown in **figure 1.1** below.

Category Group
label (ฉลาก)
fungus (รา)

Figure 1.1 Category group

Therefore, ThaiBev with the cooperation of CMKL University is trying make a system that would help categorizing bottles by adopting computer vision in their screening process. For the screening process, large datasets are required to train the system. The process of acquiring accurate dataset is labor intensive process as thousands of images needs to be labelled by experts every week/month. So, we are trying to automate the process of creating an accurate labeled dataset which will help decrease the time it takes to create training dataset which can be utilized by the machine screening process.

The overall process can be summarized by the **figure 1.2** shown below.

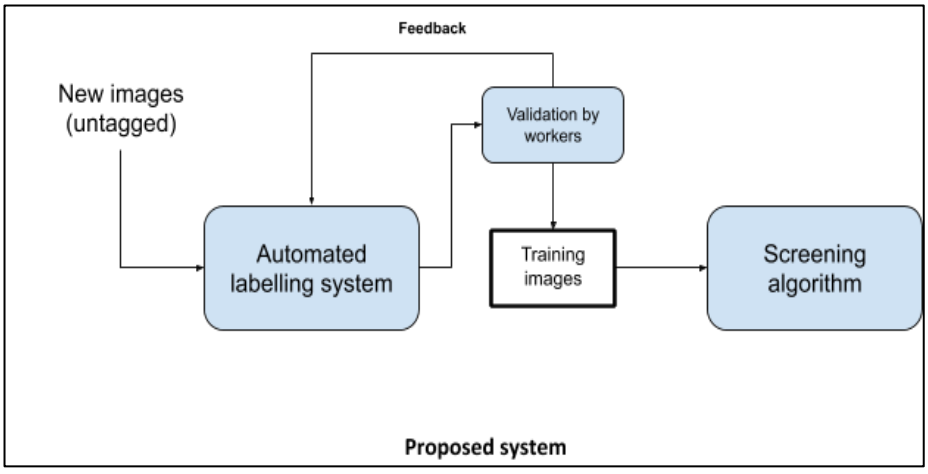


Figure 1.2 Proposed System

1.2 Objective

The process of labelling thousands of unlabeled images requires a lot of time and effort. Certain steps are taken to decrease the complexity of this task. For instance, images can be clustered before labelling the images. By doing so, we assume that each cluster will contain similar tags, whereas different clusters will contain different tags. If the previous statement is true, then the complexity of labelling (for the experts) greatly decreases, since the experts will just need to look at few images in each cluster to determine the tags contained in it.

Our main goal is to automate as many tasks as possible in the data labelling process. The purpose behind automating such a task is to decrease the time it takes to label images. While speeding up the process of labelling, we also need to maintain and ensure high accuracy in the dataset.

We have broken down the labelling process into the following categories:

- Labelling images (the actual process of labelling the images)
- Preparing the dataset (this step involves preprocessing dataset and/or storing information about the images in the dataset for further usage)
- Managing tagged data (this step involves managing the tags given by the labelers)

Each category listed above contains tasks and processes which we need to either automate or improve.

1.3 Scope

1. Preprocessing tools (data cleansing):
    - Detecting bad top and bottom-view images
    - Cropping out the center bottle in side-view images
2. Tracking user inconsistency in the data labelling process
3. Automating the processes involved in the dataset creation and labelling process.
4. Create an interface that allows for:

- Easier and convenient methodology to experiment with different clustering algorithms on a given dataset.
- More focused and featured tagging process (For instance: showing similar images within a cluster while tagging).

5. Deploying application

1.4 Method

1. Preprocessing:
    - Utilizing digital image processing tools, for instance: edge detection and Hough transforms, and machine learning techniques to pre-process images.
    - Utilizing DNN models such as YOLO for side view image bottle detection.
2. Desktop application that provides an interface for tagging images as well as experiment with different clustering algorithms on a given dataset.
3. Change layout and position of UI elements and test improvements via survey of application satisfaction rate methodology.
    - With the amount of time taken to complete a task (using our application) being the main comparison factor
4. Saving feature, tag prediction and other data for usage during labelling process.
5. Using tag data, stored locally and on Google Drive, of a given dataset to detect tag conflicts between users.

1.5 Expected Outcomes.

By the end of this project, we will have achieved the following:

1. Automate steps involved in the labelling process.
2. Tags recommending for the users.
3. Bad/garbage images elimination.
4. Allow the users to use or create their own pipeline for clustering
5. After labelling, structure your directory using the application.

1.6 Table of Operation

Operating time is demonstrated in **Table 1.1**

Table 1.1: Operation time

Task	Month								
	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May
Learn about the manual labeling to understand its workflow									
Build/ Develop the first simple version application for automating task									
Working on configuration files creation for users and bad images elimination									
Implementing and designing the early version of complete system for use									
System Experiment and Evaluation • Speed up process on tagging/re-tagging image. • Speed up model creation process.									

• Improving model creation process • Improving tagging/re-tagging image • User survey/feedback from application									
• Feedback, analysis and improving algorithms. (Accuracy and efficiency). •Improving UI/UX of application									
System demonstration and evaluation.									

1.7 Report Outline

The rest of this report is organized as follows:

Chapter 2 reviews the literature in bottle labeling system

Chapter 3 describes the design and implementation of bottle labeling system.

Chapter 4 demonstrates algorithm of bad bottom, top and side view image bottle detection, and human-assisted labeling application.

Chapter 5 closes the report, reviewing the work undertaken and draws conclusions about key parts of the work that was undertaken.

CHAPTER 2
REVIEW OF LITERATURE

This chapter discusses the literature of bottle labeling system considered during the analysis and design phase of this project. The investigation served n purposes: firstly, secondly, we wished to establish Theoretical background for labeling application that used in this project (section 2.1), Methods used for identifying garbage images (section 2.2) and showing the related work (section 2.3). Finally, section 2.4 summarizes the chapter.

2.1 Theoretical Background for labeling application

For implementation of this project, many methods and tools were used for development in labeling application. They have been separated into 3 parts, which is Methods for Front-end, Methods for Back-end, and Methods for Packaging application.

2.1.1 Methods for Front-end

2.1.1.1 PyQt

PyQt is Python library for multi-platform GUI toolkit development, which contains modules for help support developer to develop their GUI [2]. Our desktop application is based on PyQt as **figure 2.1** shown below because PyQt is supported multi-platform (Windows, Linux and MacOS) and can be used in Object Oriented Approach. Furthermore, we selected PyQt over other Python GUI toolkit such as Tkinter because of its extensive APIs and functionalities.

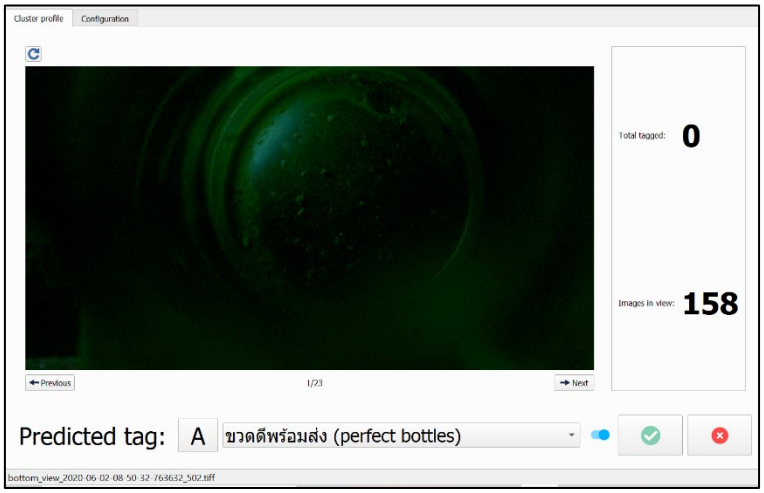


Figure 2.1 PyQt user interface

2.1.2 Methods for Back-end

2.1.2.1 SQLite

SQLite is multi-platform database, which its design can be used in Python with simple functions calls. We selected SQLite for our application's database because of its resilience and compatible with framework of PyQt. On the other hand, compared with other database management systems, SQLite is database that embedded into the target program (embedded database), while other databases usually are client to server database [3]. **Figure 2.2** below shows connection with SQLite database can be established using PyQt in as many as two lines.

```
db = QtSql.QSqlDatabase.addDatabase('QSQLITE')
db.setDatabaseName('sports.db')
```

Figure 2.2 Showing code for connection

2.1.2.2 Google Drive

Google Drive is one of the most popular cloud storages. With this advantage, it is easier to get along well with its functionality and usability. We chose

Google Drive because it has GoogleAPIclient as Python library, which meet our expectation for us to develop it with PyQt framework [4]. GoogleAPIclient modules can be established by declared its modules / libraries and import functions as we want to be used as shown in **figure 2.3** below.

```
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from googleapiclient.http import MediaIoBaseDownload, MediaFileUpload
```

Figure 2.3 GoogleAPIclient Python modules

2.1.3 Methods for Packaging application

2.1.3.1 Nuitka

Nuitka is S2S complier, which complies python to standalone executable file that can run without Python installed on computer [5]. We chose Nuitka against another Python complier such as Pyinstaller because the other complier usually freezes and package the Python file, which make the file itself unnecessary dramatically huge and the run-time performance is quite slow. On the other hand, Nuitka reconstruct the Python file to C program and connect to libpython to make it executable [5], which making its size and runtime performance close to actual Python file rather than the method that we mentioned above. From **figure 2.4** shows below size of EXE file after converting from Nuitka, which is around 0.67 MB.

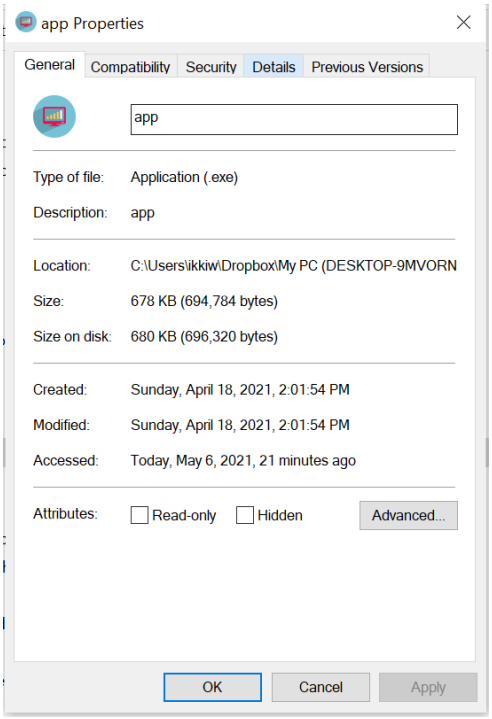


Figure 2.4 EXE file after converting Python file by Nuitka

2.1.3.2 Advance Installer

Advanced Installer is the authoring Windows installer to use to create installer for Windows [6]. Our labeling application is used for Window10 as **figure 2.5** shown below, so Advance Installer gives us the better optimization and modification among the other installer such as Inno setup and NSIS because we can place updater inside to the installer itself as **figure 2.6** shown below and generate multiple shortcuts after installation is done, so we think that Advance Installer is most suitable installer for user.

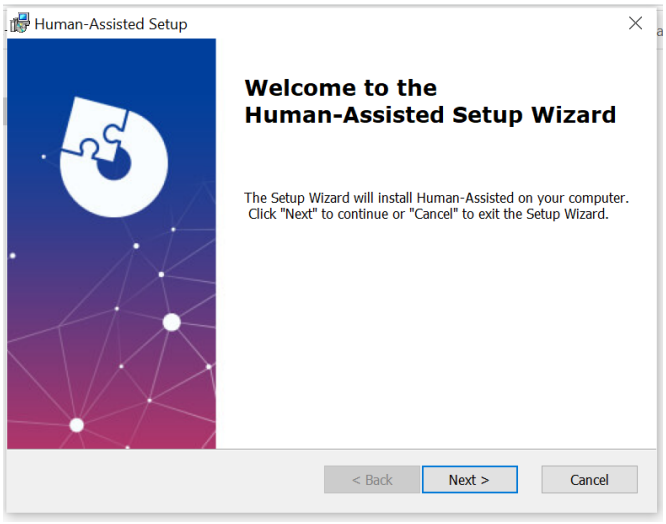


Figure 2.5 Installer

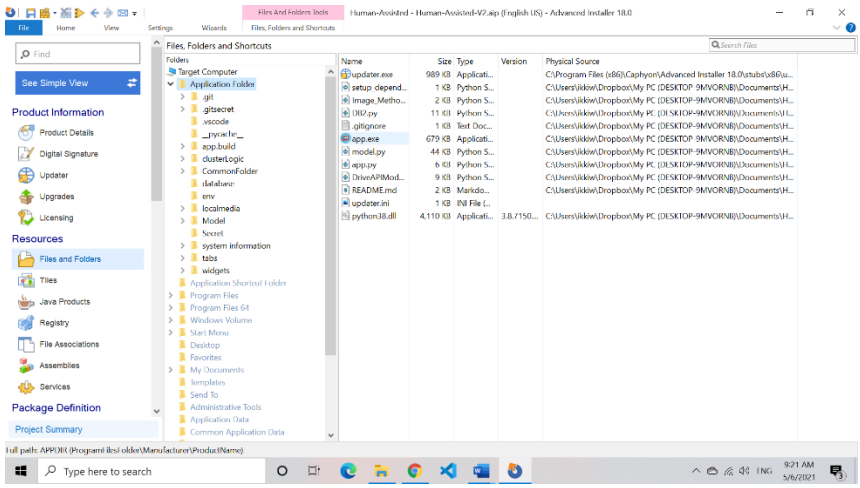


Figure 2.6 Advance Installer Interface

2.2 Methods used for Identifying Garbage Images

2.2.1 Gaussian Blur

Gaussian Blur or Smoothing is one of the techniques that blurring an image to reduce its detail and noise within the image [7]. This technique is widely used in image processing for noise removing, smoothing and reflection an image. We have

used Gaussian blur to eliminate unnecessary noise from our images. The Gaussian kernel is shown in **figure 2.7** below.

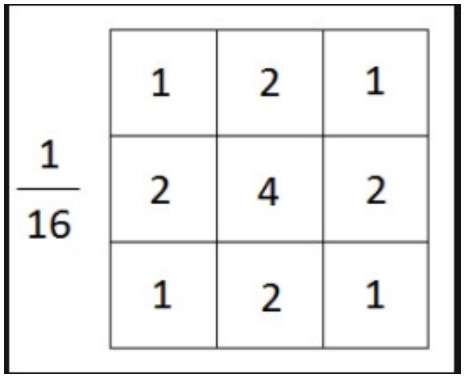


Figure 2.7 Gaussian Blur Kernel

2.2.2 Sobel Filter

Sobel Filter or Sobel-Feldman operator is image processing and computer vision technique for emphasizing the edge of the image. Its algorithm works by calculating the image's gradient intensity at each pixel within image as **figure 2.8** shown below.

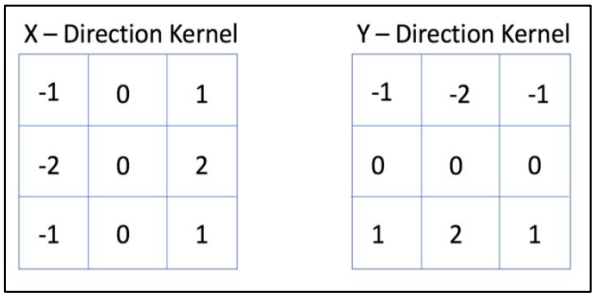


Figure 2.8 Sobel Kernel in X and Y direction

2.2.3 Circle Hough Transform

Circle Hough Transform is common feature extraction technique that used in many purposes such as computer vision, image analysis, and digital image processing. For detecting circles in an image as **figure 2.9** shown below. Hough Transform procedure is to find the instance of object, which is quite imperfect and voting

procedure to find its parameter space. [8]. The idea of Hough transform is that every edge point in the edge map is transformed to all possible lines that could pass through that point. [9]

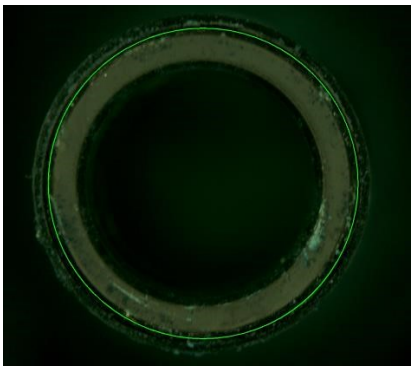


Figure 2.9 Image using Hough Transform

2.2.4 Median Blur

Median blur is the techniques that remove noise (grove / scratch and salt and pepper noise) from images such as edge detection in the image. The different from Gaussian blur filter is the median blur filter calculate the median as **figure 2.10** shown below, any pixel that is not represent in nearby area will not affect the result. [10] This filter is especially helpful for salt and pepper type noises, where there exists sharp disturbance in the image.

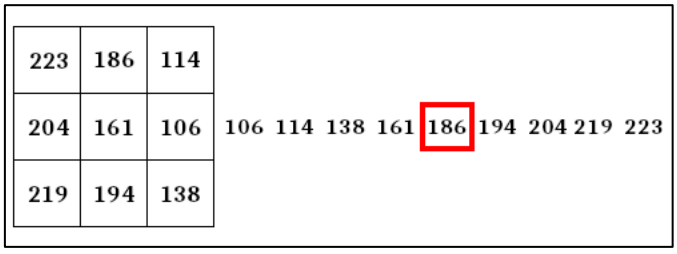


Figure 2.10 Median Blur Kernel

2.2.5 Yolov5

For detecting side view bottles, we use Yolov5 model that was developed by ultralytics. The Yolov5 model has four other versions that are named Yolov5s, Yolov5m, Yolov5x and Yolov5l as **figure 2.11** shown below.[11]

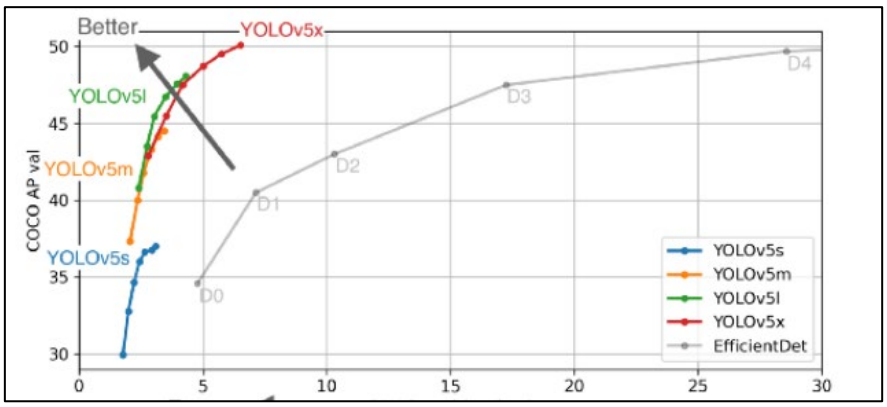


Figure 2.11 Versions of Yolov5 [11]

2.2.6 Silhouette Clustering

Silhouette clustering is the method for finding the most suitable cluster and interpretation, also it is used for validation of consistency in clusters. By computing silhouette coefficients and measuring similarity of its own cluster compared to the others as **figure 2.12** shown below. [12]

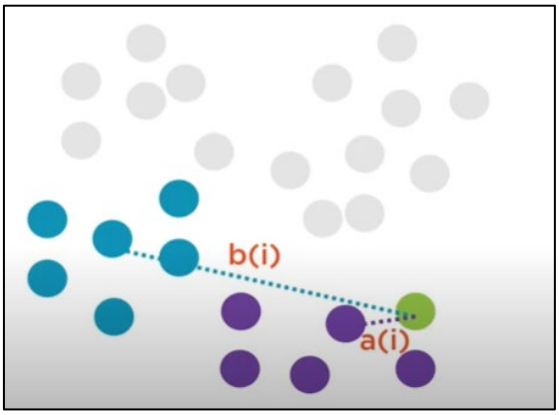


Figure 2.12 Representation of silhouette clustering [12]

2.2.7 Cosine similarity measure

Cosine similarity measure is used for detecting similarity of documents without considering their scale. By measuring cosine angle of 2 vectors lay on multi-dimensional space by considering smaller the angle, higher the similarity as **figure 2.13** shown its formula for its calculation below [13].

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Figure 2.13 Cosine similarity formula

2.2.8 LabelImg

The LabelImg is the labeling tool for image detection classification (using object boundaries box) that developed by Python languages with PyQt5 as its graphic interface as **figure 2.14** shown below. It is supported annotations that we used, which is YOLO [14].

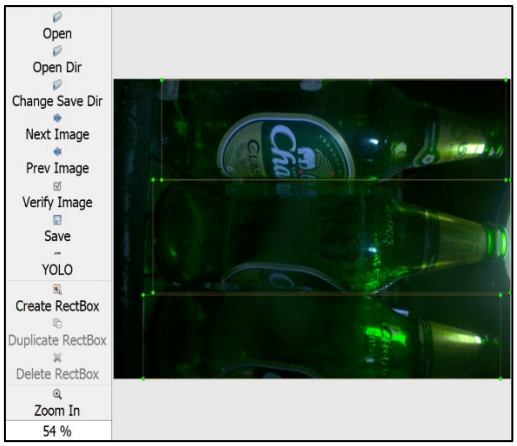


Figure 2.14 LabelImg Graphic Interface

2.3 Related work

Joris Guerin proposed combining deep feature extraction using pre-trained convolutional neural network or CNN ImageNet and clustering algorithm to

classify set of images in the labelling process [15]. They compared Keras [16] implementation of ResNet, Inception, VGG16, VGG19 and Xception with the pretrained weights. For the clustering algorithm, Scikit learn implementation [17] of K means and Minibatch K Means was used. They applied this pipeline to a robot that is equipped with a camera to sort objects as shown in **figure 2.15** below.

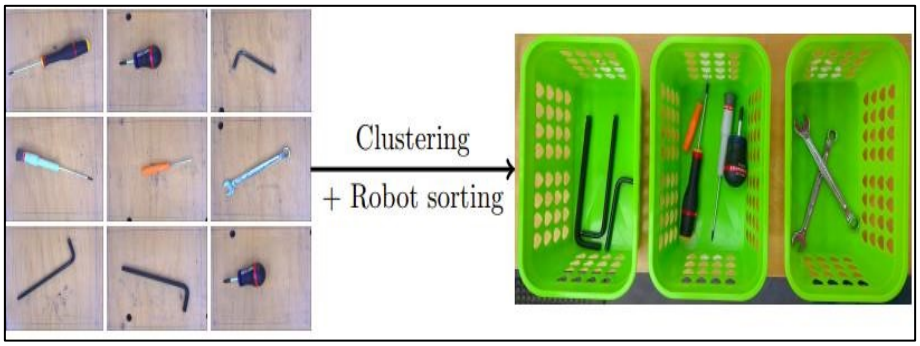


Figure 2.15 Result of Robot Sorting

There exist certain crucial differences between our problem domain and the problem domain dealt with in the paper (COIL100 and VOC2003; dataset used in the paper). The assumption that they made in the paper is that the number of clusters is known whereas in the bottle labelling problem, the number of clusters are not known because it is difficult to know what kind of dirtiness, scratch or trash the bottle possess and every time we see things in the bottle that was not seen before. Since we categorize bottles condition whether it is in suitable condition for recycling process.

The next related work is DASSL: Dynamic, AI-assisted, Scalable System for Labelling Used Bottle Images [18]. The aspect of this system which relates to our project is the usage of clustering and feature extraction in order to speed up the labelling process as well as improve consistency. Where the paper states a reduction of labelling time by at least 10x when clustering and feature extraction is utilized. Our application essentially provides an interface and automation feature for the

clustering and feature extraction described in the paper as well as other aspects of the labelling process.

2.4 Chapter Summary

In Chapter 1 we proposed the project's background, scope, method, objective, expected outcomes and overall plan for the entire project throughout semester 1 and 2.

In this chapter the state-of-the-art was categorized into Tool used (section 2.1). Observations were made on the systems reviewed Methods used for identifying garbage images (section 2.2), and Related work (section 2.3). The relevance of the literature review to overall bottle labeling system was summarized (section 2.4).

The next chapter presents the design of algorithm of detecting bad images in different view of bottle and application for bottle labeling system, which is a system intended to decreasing time for classification bottle for recycling.

CHAPTER 3
METHODOLOGY

3.1 Introduction

In Chapter 2 we reviewed the overall literature of bottle labeling system project. This chapter describes the design of labeling application, a system that helps labelers to tagging the bottle by browsing bottle image into the application and tagging the condition of the bottle. Also, the design of algorithm to detecting bad bottle in multiple views, which is side, bottom, and top views. First, the chapter describes the overall project's system design (section 3.2), and storage system in this project (section 3.3). Method for detecting bad images is then discussed (section 3.4), and miscellaneous is mentioned (section 3.5). Lastly, chapter 3 is summarized in section 3.6.

3.1.1 Image properties and labelling process

Currently, the main project focuses on beer bottle images. The images are taken in 3 different angles: top, bottom and side (which includes left and right) view. At the time of writing, there exists around 70 image tags, which are grouped in 3 categories (referred to as "tag class"):

- A : The bottle can be sent directly to the appropriate cleaning station.
- B : The bottle contains items or properties that requires human supervision before it is sent to the cleaning station.
- C : The bottle has properties that cannot be removed or fixed by supervision or the cleaning station. It cannot be recycled and will get crushed.

On a monthly basis, un-tagged images are collected from the factory and the experts or labelers are required to label them. The labelling process usually requires the labelers to visit the university, and each labelling session has a certain requirement. For instance, on certain days, only C tag class is required for side view.

The current process of labelling involves a set of directories (each representing a cluster) and the labelers are required to surf through these directories and move them

to a specified directory (where the name of the directory represents the image tag). During labelling, the labelers might come across bad images, images with new tags, etcetera. This process involves many repeated steps that can be automated.

3.2 Application and features

Our application focuses on automating as many steps as possible in the **labelling**, **preparing the dataset** and **managing the tagged images process**. Our application provides the following automation features for each of the mentioned processes:

- Labelling
    - Once an image is tagged, the application groups together similar image, in the same cluster, together. Purpose of this step is explained in the (discussed in section 3.5.1).
    - Focusing on only certain tag class. This is done with the help of using the main model to predict the tag class of a given image.
    - For users, who are familiar with clustering, the application allows a simple interface to cluster (using any of the available scikit-learn clustering algorithm) the images in the directory.
    - Bad images (discussed in section 3.4) are from the dataset prior to tagging. Therefore, allowing the labelers to waste less time labelling these images.
    - Users are not required to move images from directory to directory. Tagged information are stored in the database (discussed section 3.3.1) and images are tagged by clicking the tag button.
- Creating dataset
    - The tag class of the images are predicted using the current available model (discussed section 3.3.1).
    - The dataset is automatically clustered. This step utilizes K-means as the clustering algorithm and tries find the best number of clusters between 2 and 50 (discussed in section 3.5.3).

- Image feature is extracted (given a trained model from Keras) and stored for further use (during the labelling process; Specifically in the clustering and closest image step). This is especially helpful, since the feature extraction step takes significant time, and since the data is stored, the image need not be re-extracted for clustering with different parameters (discussed in section 3.3.2).

- Managing tagged images
  - Since the image tags are stored in the database and each tag is associated with a user. Due to this the user can revisit previous tagging sessions and continue tagging.
  - Once tagging is complete, this information is automatically converted into spreadsheet and stored locally, as well as on Google Drive.
  - Given all the tagging information, the users can use the application to find conflict in tags given to image between users (discussed in section 3.5.2).

3.2.1 System architecture

Figure 3.1 describes the architecture of the labeling application by in the local, the labeling application will receive image data, image feature data and local database from the dataset or directory for usage in tagging method. By that, whenever tagging method is finished by user, application will automatically upload the tagged data into Google Drive. Certain information such as saved parameters for model and tag list is stored in Google Drive and the application will automatically or upon request download them for further usage in the application. The application stores dataset information on Google Drive, as well as receives dataset information from Google Drive and stores it in the dataset. The set of data that stays local are the images, feature data and certain local database information (session information, etcetera).

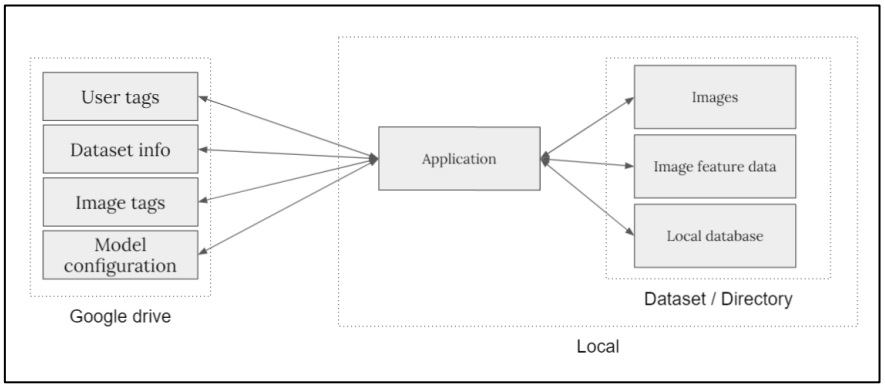


Figure 3.1 Architecture of the labeling application

Figure 3.2 portrays the architecture of dataset that used in the application. This architecture shows the folder and file inside the dataset after unprocessed dataset is pass through creating dataset process by adding necessary folder and file into dataset, which is CSV user tags, Image feature data, good images, bad images as folder and local.db, info.txt, and .mats as single file for use in the application.

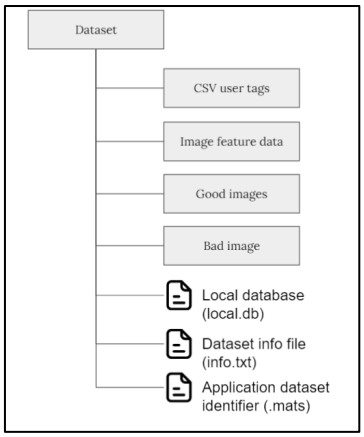


Figure 3.2 Architecture of dataset that is used in the labeling application

3.2.2 Use case and sequence diagram of the application

3.2.2.1 Use case diagram of Labelers

As shown on **figure 3.3**, labelers can start using the application by browse the image dataset by when they are opening dataset, they can request focus directory to cluster to better focus to the dataset. If they already have name exist in session, they can either choose between resume from their last session or cancel the resume session. For create tag and search, it can be available for use by changing mode to all tags/ classes mode by they can create non-existence tag and add it into the application, and they can search to find any exist tag that available in application. Next, it is change view and cluster in the dataset by users can choose image group and bottle view. For tag image and end tag session, users can choose tag that suitable for bottle condition and tag it by either press keyboard Enter/Return button or green button in the application to tag it, and it will automatically find the closet image from the dataset that is used in the application with re-order cluster to making labelers work faster. When labelers tag wrong images or make mistake, they can press Ctrl-Z to undo the tagging method, and the application will return to previous stage to let users tag it again. After tagging session is done, application will convert tag data into CSV file and store it in Google Drive. Lastly, labelers can check availability for update. If there is an update available, labelers can update the application to its latest version.

Figure 3.3 Labelers Use case diagram

3.2.2.2 Use case diagram of expertise

For expertise as shown in **figure 3.4**, there will be additional features that add from labelers, which is cluster dataset, which expertise can cluster the images in its dataset, and create dataset, which is contains features for helping expertise to deliver dataset to labelers by once the create dataset is used, it will create Google Drive directory and database for each dataset, also it included feature extraction from ResNet 101,152, and 50. Expertise can also zipping dataset/ directory as zip file, which include all feature of create dataset that we mentioned within target directory/dataset to deliver to functionality to help support labelers to label/tag faster.

Figure 3.4 Expertise Use case diagram

3.2.2.3 Application sequence diagram

The application sequence diagrams shown below consist of 3 main features of creating, opening, and tagging dataset, which each of them contains feature that help users to tag image faster:

- Creating dataset
    - Create Google Drive for CSV storage
    - Create database and add images to database
    - Create predicted image class and store it in database
    - Create signature file into directory
- Opening dataset
    - Resume session for existence name / Create new session for non-existence name
    - Load image and tags data
- Tagging dataset
    - Insert tag data into database
    - Switch cluster and re-order images
    - Convert tag data into CSV and store to Google Drive
    - Undo tag for the mistaken tagged image

❖ **Opening dataset Sequence diagram**

According from **figure 3.5** shown below we can see that when the user comes and browse image dataset for tagging bottle condition images, after they are done browsing images, there will be resume session dialogue showing in the application for input name. When the user inputs their name, there will be 2 paths in application. The first path is name for resume session is existed, so database will validate user data and show dialogue for resuming session to user. On the other hand, if there is no session name existed in database, the database itself will store new session name to itself and waiting to load the existed name again. After this process is done, the application will request a load image and data from database and send both image and tags into application to setup data structure, and once this process is done, the application will show image to the user.

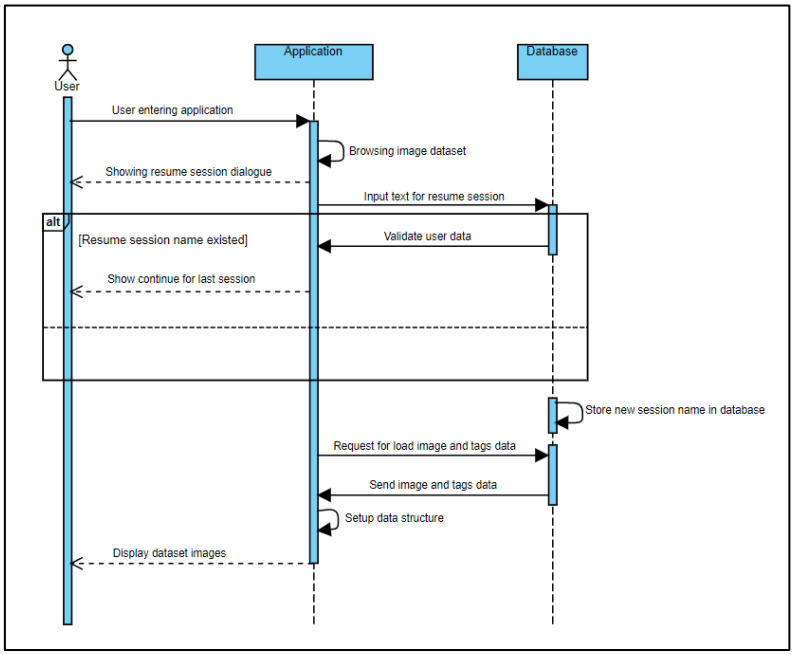


Figure 3.5 Opening dataset Sequence diagram

❖ **Tagging dataset Sequence diagram**

We can see that **figure 3.6** that shown below when the user tags the image, application will send tag data to database to store selected tag with image, and application will check cluster that there is untagged image left or not. If there no images left them application will switch finished cluster to other cluster that available in dataset. Also, if user tagged wrong tag, there is undo tag by pressing Ctrl-Z to reverse stage from previous image and tag that user did and display the same image that user did wrong again. If the tag process is correct and there are still images left in cluster, the application will find the closet image of the according to previous image that has been tagged. Application also requests image data from database and after the request is sent, application will receive the images data from database. After that predicted tag and UI for new image, and user will repeat the process again as we mentioned above until user finished tagging session, so the user will need to press end tag session, the application will show dialogue to guarantee to double check user for ending session, after that application will send request to database for receive all tagged data and covert

it to CSV and using dataset ID for Google Drive storage. Lastly, the application clear UI and remind user.

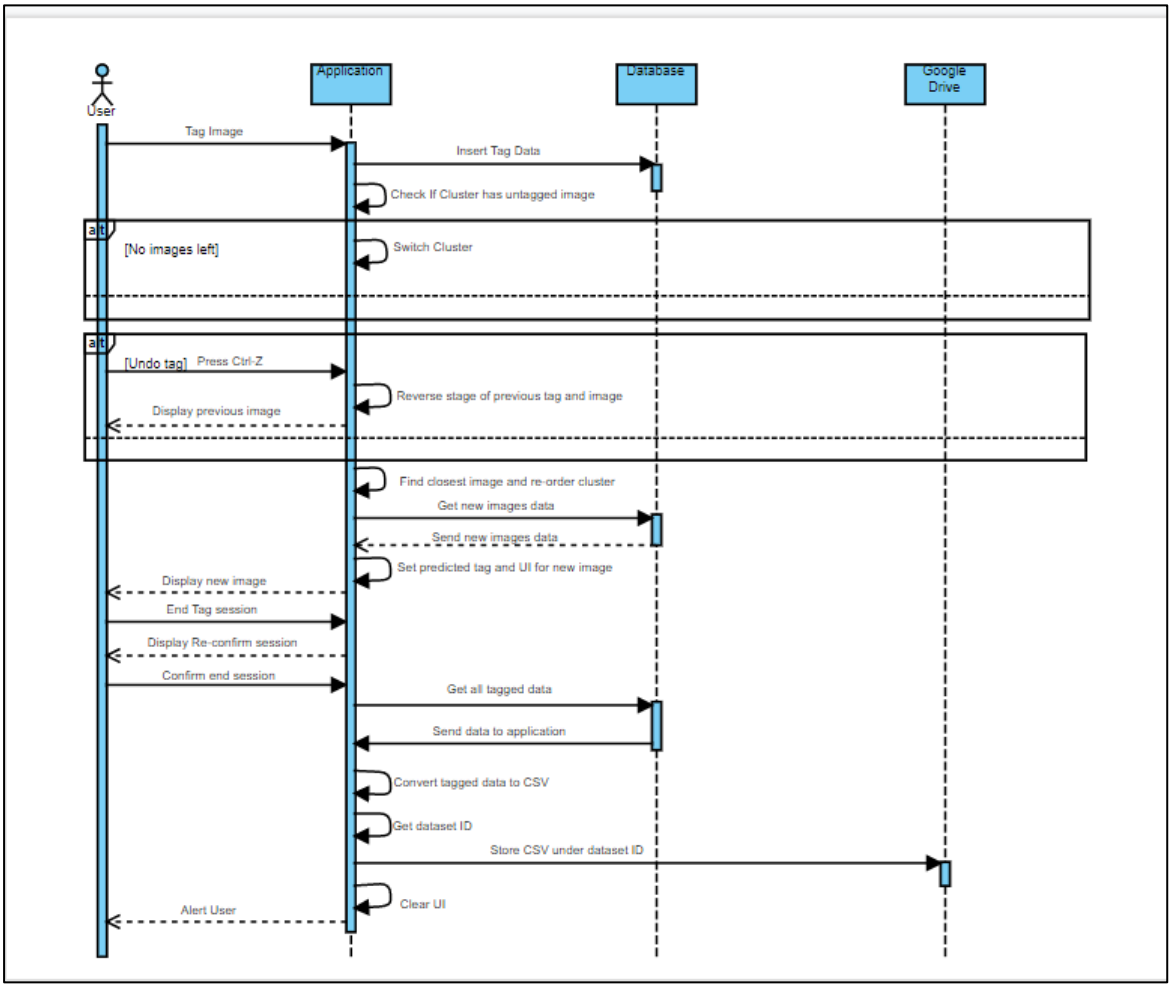


Figure 3.6 Tagging images Sequence diagram

❖ **Creating dataset Sequence diagram**

From **figure 3.7** shown below, user will be required to select dataset directory, choosing pre-trained model (ResNet 101,152,50), and prediction configuration (bottom, side, top, right, and left views). After that application will collect images, remove bad images of dataset, create database, and add those images into database. The application will add signature file into the directory and predicted image class and store

the predicted image class into database. Next, the application will proceed feature extraction, store it, and find the best cluster amount on dataset for k-mean and then cluster the directory. Finally, the application will create a directory with hash name and clear any directory and open file explorer to user to check the dataset.

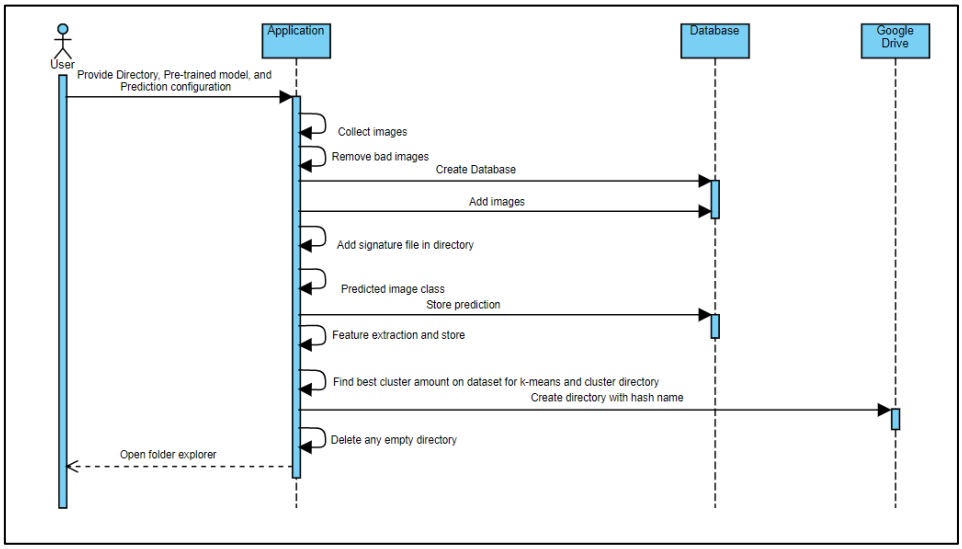


Figure 3.7 Creating dataset Sequence diagram

3.3 Storage system

3.3.1 Database implementation.

The database created exists in the dataset directory. It contains essential information about the images and is generated when the user requests the application to convert an image directory into a dataset. The database contains the following tables:

- Image name

Table name: **Dataset_Image**

Table 3.1: Dataset_image database

Attribute	Datatype	Description	Primary/Foreign key
ID	INT	Image id	Primary key
Image	TEXT	Image filename	

- Image tags

The image tags are maintained in a CSV file stored in Google Drive. The following table contains the same information as the file in Google Drive. The purpose of having a local table is to utilize the tag information locally as well as due to users being able to create new tags while tagging images. The newly created tag should not be added in the main tag file on Google Drive, therefore this tag is added in the local db. Whether to add the newly created tag should be added in the main tag file can then be discussed by the labelers and experts.

Table name: **Tag_info**

Table 3.2: Tag_info database

Attribute	Datatype	Description	Primary/Foreign key
TagID	INT	Tag id	Primary key
TagName	TEXT	Tag name	
TagClass	TEXT	Class the tag belongs to (A,B,C..)	

- Model prediction of each image

The application utilizes the currently used model to predict the image class (A, B or C). For each image, the model gives a probability distribution of each class. The class with maximum probability is chosen as the predicted class. Given this probability distribution, we can estimate the confidence of the prediction. In this case we used "margin of confidence" measure:

$$Confidence(prediction\ distribution) = 1 - (P_{highest} - P_{second\ highest})$$

This confidence calculates the difference between the class with the highest and second highest probability. If the highest probability value is very close to 1 (i.e., model is confident in the prediction) the confidence score will be close to 0. Confidence values as such can help pinpoint current model's weakness as well can be utilized in the tagging process.

Table name: **Image_Prediction**

Table 3.3: Image_Prediction database

Attribute	Datatype	Description	Primary/Foreign key
imgID	INT	Image ID	Primary & Foreign key
tagID	INT	Tag ID	Foreign key
Margin_confidence	REAL	Model's confidence on the given prediction	

- Information about the tagging session

This table is used to track the tagging session. Each tagging session is associated to a user. A tagging session can be used to track the images that were tagged by the given user and also be used to resume the tagging session.

Table name: **Tagging_session**

Table 3.4: Tagging_session database

Attribute	Datatype	Description	Primary/Foreign key
sessionID	INT	An identifier for the given tagging session	Primary key
user	TEXT	User who initiated this session.	
time_started	REAL	The time this session was initiated. (Stored in unix time)	

- Information about tagged image.

This table tracks the tagged image in each tagging session. The image tag is in a string format. Given the tag table, each tag is given a unique identifier. The tag string contains zeros and ones. The index of the tag string is the tag identifier. Therefore, if the image has tag ID 3, then the third index of the tag string will be one. Example of a tag string: "00010…000". Using this format, if image tags change, only the row needs to be updated and if new tags are created the tag string size will increase. This saves space, compared to having an image and tag pair has a single row, and insertion/update

time. Having tags as columns would not be a good solution either since users can create their own tags, requiring the creation of a new table with updated columns (this would also require the insertion of all old data in the updated table).

Table name: **Tagged_image**

Table 3.5: Tagged_image database

Attribute	Datatype	Description	Primary/Foreign key
sessionID	INT	Session the image was tagged in.	Primary & Foreign key
image_id	INT	The image that was tagged	Primary & Foreign key
image_tag	TEXT	Tags given to the image.	
time_tagged	REAL	Time the image was tagged. (stored in unix time)	

Figure 3.8 below shows the database relation in labeling application.

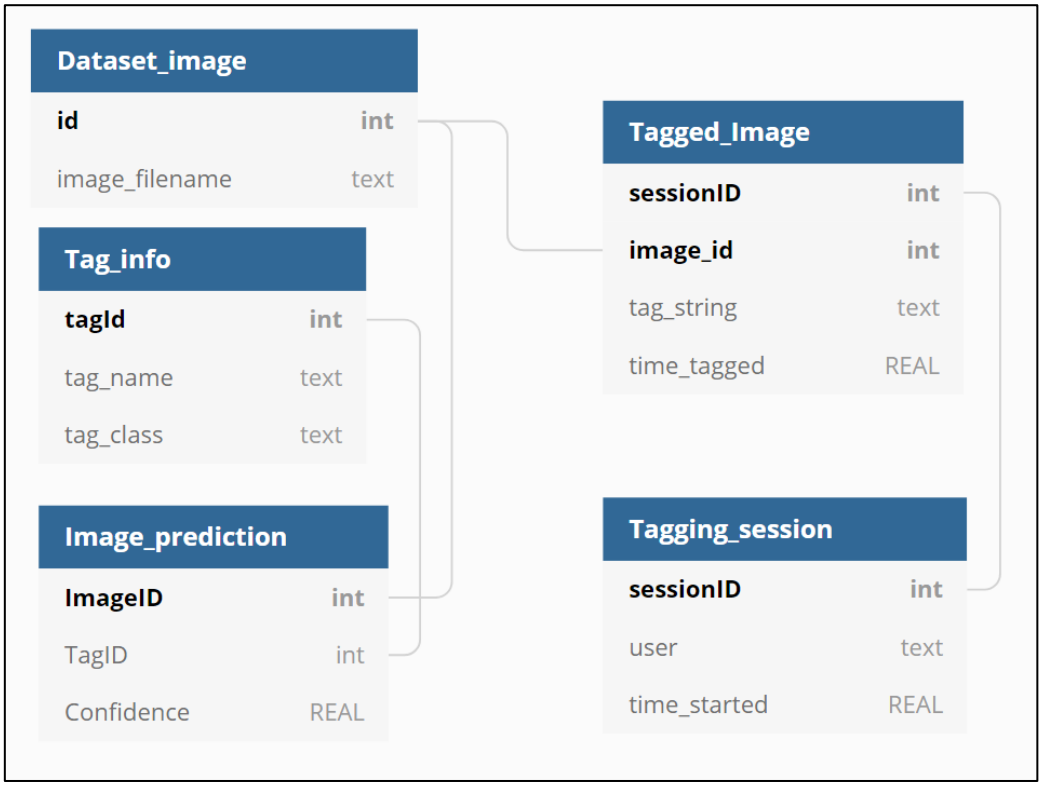


Figure 3.8 Database Relation

3.3.2 Image feature data

When creating dataset, the users can choose the desired pretrained model they would like to utilize for extracting features from an image. Some of the models include: ResNet50, ResNet101, etcetera. These features are extracted and stored under the directory named “data”. And these images are stored in a “.npy” file. Due to this we can treat the image feature data as a NumPy object. Certain thing to note here is that the features for each image contain >2000 dimensions for certain models. Given such a high dimension and huge dataset, it exceeds the available memory and causes memory leak. Due to this, we have utilized memmap [19] provided by the NumPy library. Which maps the data stored in a disk to an array. Therefore, most of the feature data is stored on disk and helps avoid creating any memory leak. To keep track of the image a feature belongs to, we store this information in another file called “image_name.npy”. When loaded, these two data are mapped to each other ({image: feature}).

3.3.3 Google Drive

Application features and methodology that involves Google Drive:

- Model configuration

As mentioned in the previous section, tag class of a given image is predicted to aid the labelling process. This prediction utilizes the current model and the saved parameters. The saved parameters are stored in Google Drive. On top of this, there exists multiple saved parameters per view, as well as the new parameters are added. The set of parameters (for all views) are separated by weeks as **figure 3.9** shown below.

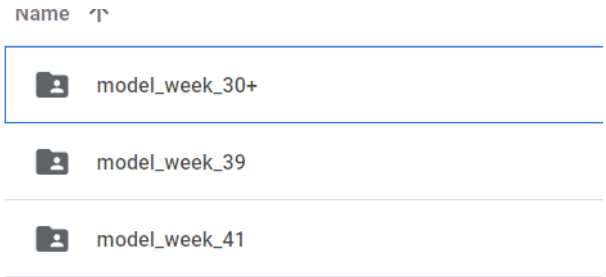
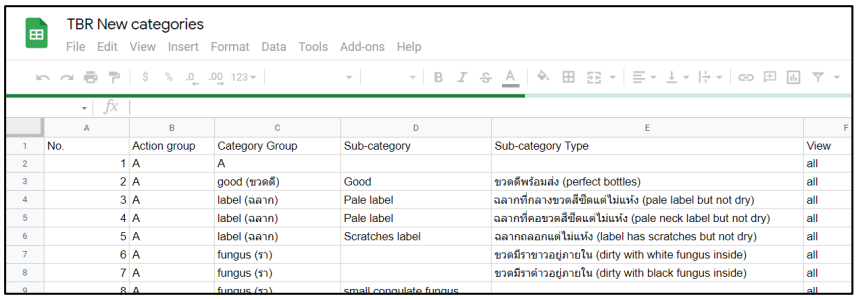


Figure 3.9 Model Configuration

The application, when opened, automatically downloads the latest saved parameters for each view. Then the updated saved parameters can be utilized to predict tag class of images.

- Tag list

The tag list is stored in a spreadsheet on Google Drive as **figure 3.10** shown below. Upon request, our application can download the spreadsheet, when updating the dataset



	A	B	C	D	E	F
1	No.	Action group	Category Group	Sub-category	Sub-category Type	View
2	1	A	A			all
3	2	A	good (ขวดดี)	Good	ขวดดีพร้อมส่ง (perfect bottles)	all
4	3	A	label (ฉลาก)	Pale label	ฉลากที่กลางขวดสีซีดแต่ไม่แห้ง (pale label but not dry)	all
5	4	A	label (ฉลาก)	Pale label	ฉลากที่คอขวดสีซีดแต่ไม่แห้ง (pale neck label but not dry)	all
6	5	A	label (ฉลาก)	Scratches label	ฉลากถลอกแต่ไม่แห้ง (label has scratches but not dry)	all
7	6	A	fungus (รา)		ขวดมีราขาวอยู่ภายใน (dirty with white fungus inside)	all
8	7	A	fungus (รา)		ขวดมีราดำวอยู่ภายใน (dirty with black fungus inside)	all
9	8	A	fungus (รา)	small conguiate fungus		all

Figure 3.10 TBR New Categories CSV

- Image tag data

When users complete tagging a dataset. This information is converted into spreadsheet format and uploaded to Google Drive as **figure 3.11** shown below. This allows users to track the tags given to each image by labelers, without requiring the labelers to do anything.

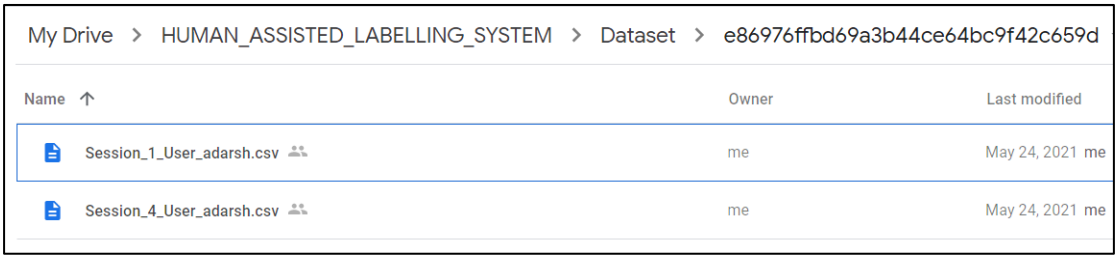


Figure 3.11 Tag data session CSV

3.4 Methods for detecting bad images

The dataset is processed and labelled before being fed into the deep learning algorithm. The processing steps include data cleaning, data augmentations and much more. For the bottle images, due to the practical requirement and setup of the cameras in the factory, certain images do not capture the entirety of the bottle/view. Due to this, these types of images cannot be labelled accurately. In this section we explain the algorithm we utilize to detect these types of bad images in the dataset. Using the algorithm described in the following paragraphs will further help automate the labelling process.

3.4.1 Bottom and top view images

Both bottom and top view images are built on similar concept. For both views, the edges are first detected, then circles are detected using Hough transform and using the result from Hough transform, our algorithm determines whether the image is good or bad. The main idea being that if a circle is not detected, the image is bad. And if it is, a final component helps determine if the image is bad as **figure 3.12** shown below.

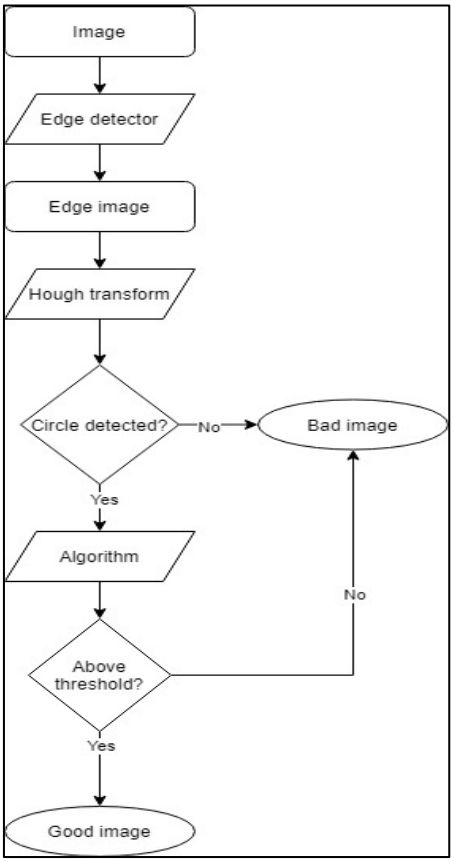


Figure 3.12 Flow chat for bottom and top bad image detection

The main idea behind detecting circles is that both top and bottom view are circular. Therefore, if we can create an algorithm that can perfectly detect the bottom/top view circles, then we can determine whether a given image is usable.

The main difference between the algorithms of the two views is in the edge detection and algorithm step (the step after circle is detected). Images of both views are taken through a camera that is set above the bottle. The camera focuses on the top of the bottle when taking image of the top view and likewise for bottom view. Due to this, to take a proper photo of the bottom view, the bottle needs to be positioned at the center of the camera. If not, parts of the bottom view will be blocked. This also implies that the item or objects that are blocking the bottom view, is the bottle itself (the **figure 3.13** given below illustrates this). These parts of the bottle have similar visual properties to the bottom view. Making it harder for our algorithm to perfectly detect the bottom view edges. The top view images have an advantage since a small shift in positions of the

bottle will not affect the top view and usually the bad top view images are due to the bottle being positioned at the edge of the camera's view. Therefore, it is harder to extract bottom view edges and requires more steps (relative to top view images).

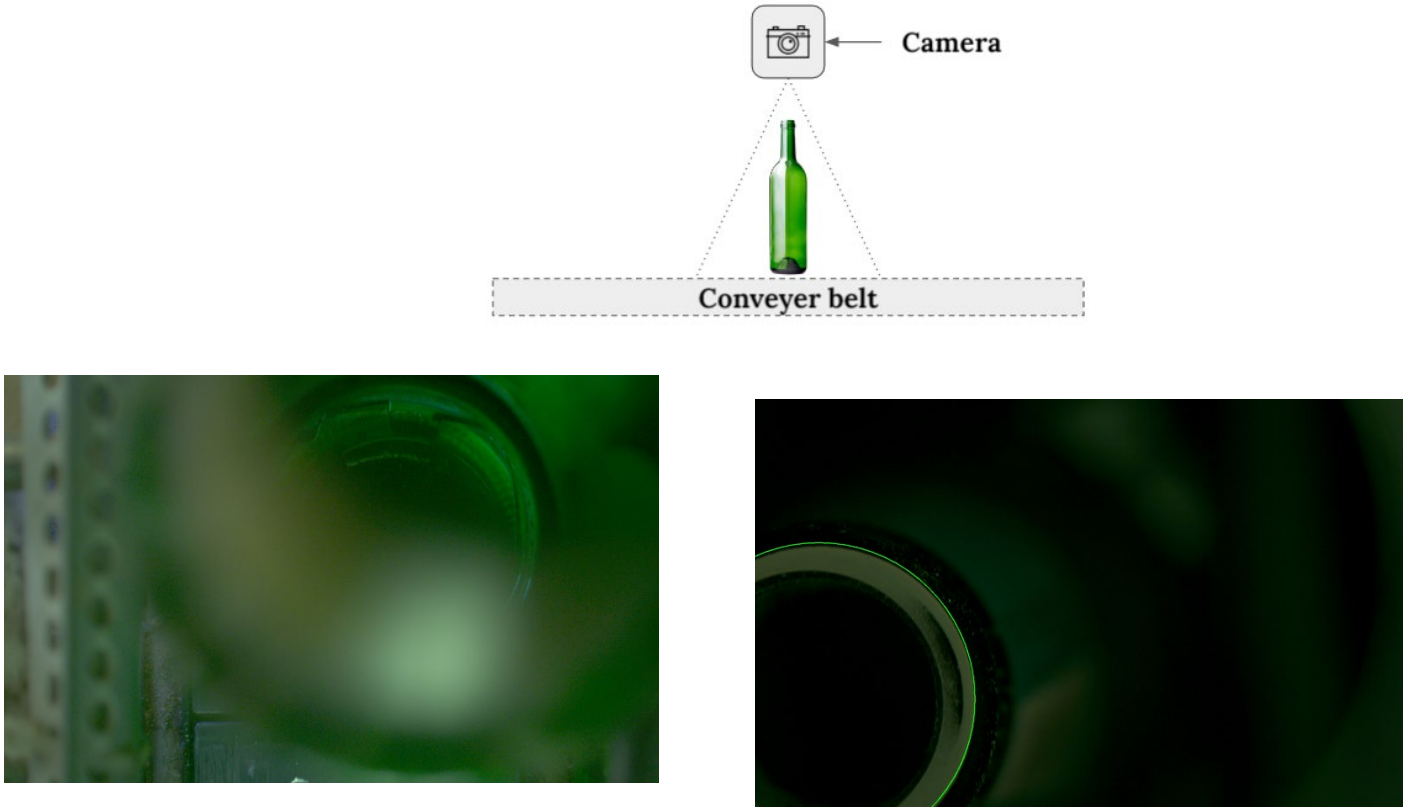


Figure 3.13 Imperfect conveyer belt within images

3.4.2 Top view images

The following bullet points, explain in detail the image processing tools and parameters used to detect bad top view image:

1. Smoothing images

   The images are usually blurred or smoothen before edges are extracted. This is done to remove any noises and spurious edges from the initial image. The exists many blurring filters. For top view image, we utilized gaussian blur with kernel size of

15. These filters were chosen after doing a cursory comparison between default values of different smoothing filters, similarly for the kernel size.

2. Extracting edges

The image is then converted into grayscale before edge extraction. For extracting edges, we used Sobel filter with filter size of 5. Again, we did cursory comparison between different edge extracting filters and found Sobel filter to be the most effective. Since this filter essentially computes the differential between neighboring pixels in a kernel, the resulting images have ranges of values. The higher the value, the stronger the edge. Therefore, we apply thresholding to the image to remove small or spurious edges. Without thresholding, the Hough circles algorithm takes significantly longer time to detect circles. And in many cases, the Hough circles detect many false positives.

3. Hough circle and detecting circles.

Using Hough circle, an image processing tool provided by OpenCV [20], we can detect circles given the edge image. This tool outputs a list of circles with their coordinates and radius. By tuning the parameters, we can further help identify bottle circles. For top view images, if the bottle circles are not detected, the image is classified as bad (as mentioned in the control flow diagram in section 3.4.1). If circle is detected, for top view, further steps are taken to determine whether the image is unusable/bad. The Hough circle tool has the following essential parameters:

- **param2** – The accumulator threshold. If high, only perfect, or almost perfect circles will be detected. If low, small curves might be detected as a circle.
- **minDist** – The minimum distance between 2 detected circle.
- **minRadius** – Minimum radius of the circle to be detected.
- **maxRadius** – Maximum radius of the circle to be detected.

For top view images, we will discuss the method through which we obtained the following parameters in the next chapter, the following values of the parameters are used:

- **param2** → 50
- **minDist** → 1000
- **minRadius** → 350 (smallest size of top view)

- **maxRadius** → 500 (largest size of top view)

4. Determining bad image

Due to the parameters used for Hough circles, the circle with the highest likelihood of being circle is output. Given the co-ordinates of this circle as **figure 3.14** shown below, the final component of the algorithm tries to detect whether the entire view is present. This is done by getting the top, left, right and bottom extremes of the circle:

a. top (mid-point + radius; y-axis)

b. left (mid-point – radius; x-axis)

c. bottom (mid-point - radius; y-axis)

d. right (mid-point + radius; x-axis)

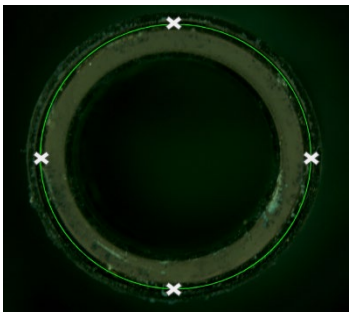


Figure 3.14 Co-ordinates of circle

After computing these points, the algorithm checks if these points are within the boundary of the image. If not, we estimate the following ratio:

$$ratio = \frac{amount\ of\ pixels\ outside\ image}{radius\ of\ the\ circle\ detected}$$

Higher the ratio, implies higher the amount of the bottle is outside the image boundary. This ratio, in our algorithm, is set as 0.4.

$$Image\ bad\ (circle) = \begin{cases} good, & ratio \leq 0.4 \\ bad, & ratio > 0.4 \end{cases}$$

Figure 3.15 below illustrates the steps involved (described above):

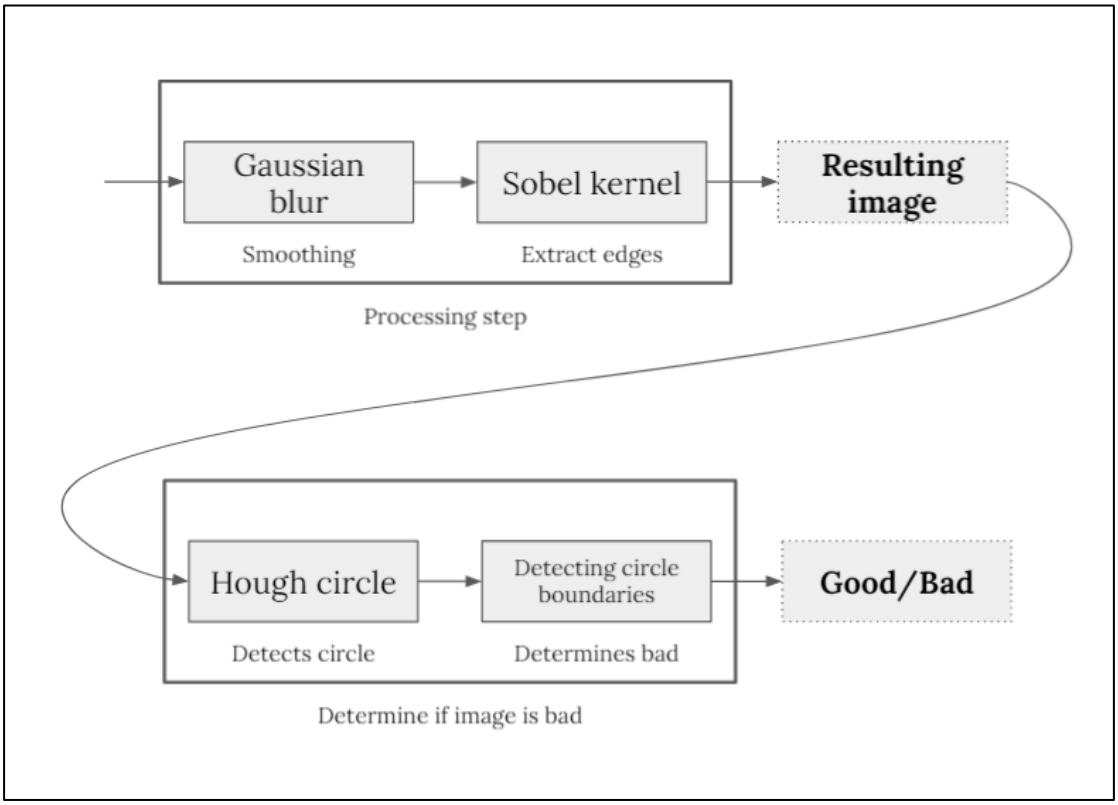


Figure 3.15 Overall process for bad top view bottle detection

3.4.3 Bottom view images

Due to the problem explained in the previous section, the bottom view image edge extracting technique requires a few more steps. The steps and their purpose are described below:

1. Using only green channel of the image

The images contain red, blue, and green channel. For top view images, the edges are extracted on the grayscale image. The grayscale image averages out the 3 channels. Therefore, the grayscale image contains the noise from all the channels. After further experimentation, we learned that if we only utilize the green-channel image, more true edges of the bottles are being detected (as shown in the **figure 3.16** comparison below).

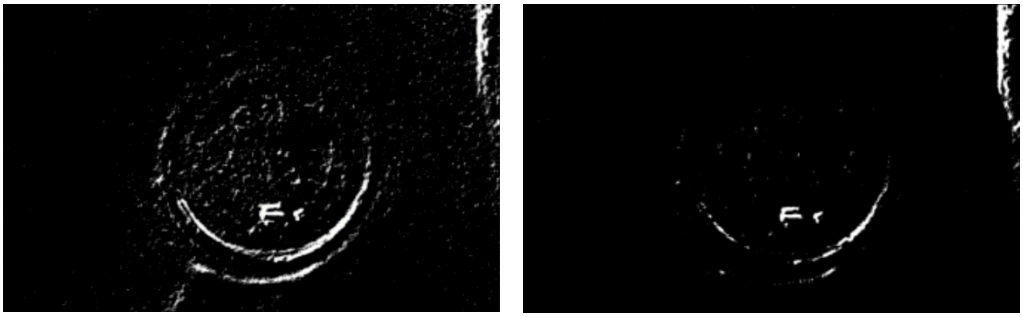


Figure 3.16 Comparison for using only green channel of the image

Due to this, for bottom view image, we only utilize the green channel of the given image.

2. Smoothing image

The green channel image will then be going through a couple of smoothing filters to denoise the image. As mentioned in top view image, gaussian blur is utilized to smooth the image. The denoised image then goes through another smoothing filter called median blur. Median blur helps remove salt and pepper type noises in image. Which essentially looks like static. As we can see **figure 3.17** below utilizing median filter further helps remove spurious edges from the image (as **figure 3.17** shown Left: without median blur; Right: with median blur).

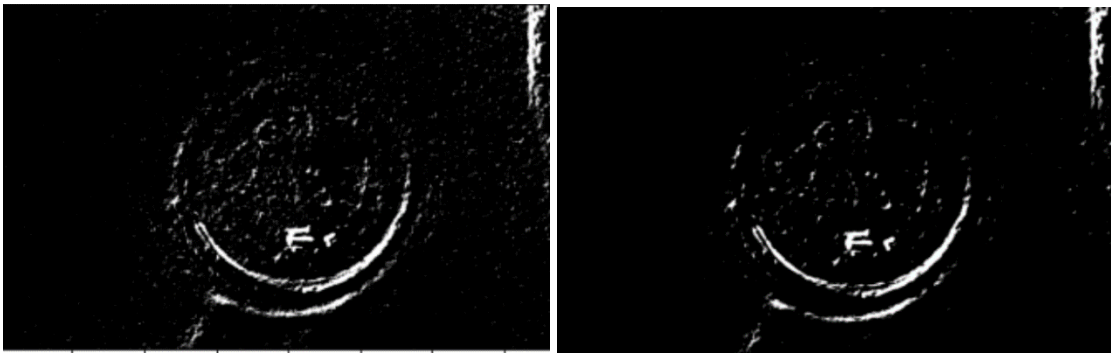


Figure 3.17 Comparison before and after smoothing image

3. Edge detection and steps following it.

Same parameters and kernel (Sobel filter) are used for detecting the image edge. But for bottom view, we use a lower thresholding value. Meaning more spurious or false positive edges will be detected. As well as true edges that have low edge magnitude. As we can see in the image below (left image), the resulting edge image (due to low thresholding) becomes substantially noisy. To remove these noises, we smooth the resulting edge image using median filter. This works due to most of the noises in the edge image appear to be salt and pepper type. By applying this filter, we can preserve the true edges with low edge magnitude, as well as significantly denoise the image as **figure 3.18** comparison shown below.

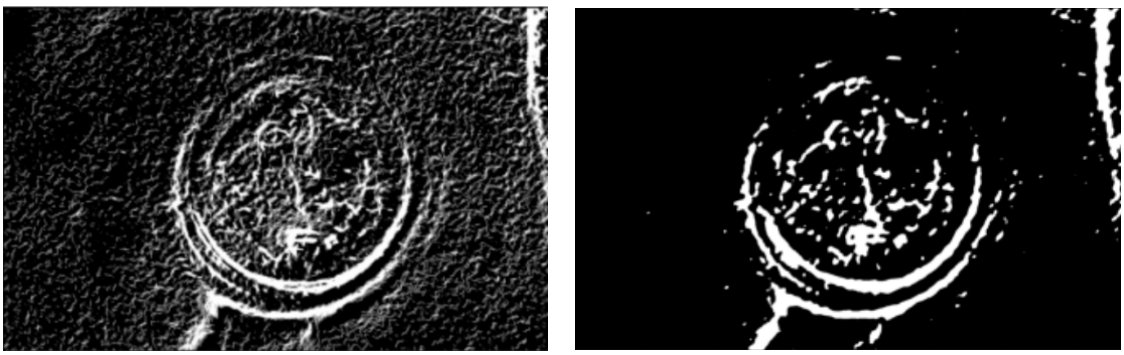


Figure 3.18 Comparison of denoise images

Finally, if we compare this method to the processing steps used in top view image, the edges extracted using this method is significantly more as **figure 3.19** comparison shown below.

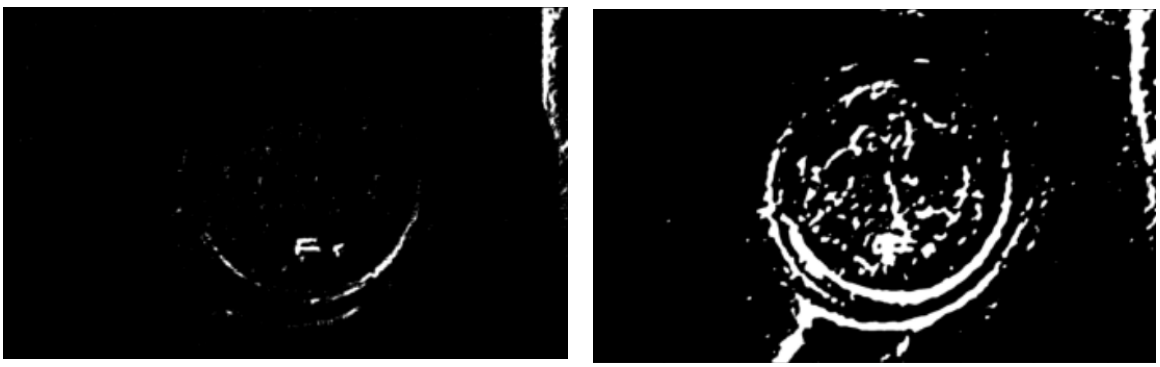


Figure 3.19 Comparison between edge detection to top view image

4. Hough circle and detecting circles.

The Hough circle step here is not modified. The only difference between top and bottom view in this step is the parameters used to detect the circles. Certain obvious parameters are modified, such as maximum and minimum radius. Even with the modified edge extraction technique, it is not guaranteed that only the bottom view edge will be detected. Due to this, we decrease the param2 parameter of Hough circle to only weed out images that barely have any circles in them.

- **param2** → 40-50 (range explained in chapter 4)
- **minDist** → 1000
- **minRadius** → 300 (smallest size of bottom view)
- **maxRadius** → 550 (largest size of bottom view)

5. Determining bad image

There exists no specialized algorithm or component for bottom view images after Hough circle. Since, if circle is detected, the image is classified as usable and vice versa if no circles are detected. The **figure 3.20** below illustrates the steps involved (described above):

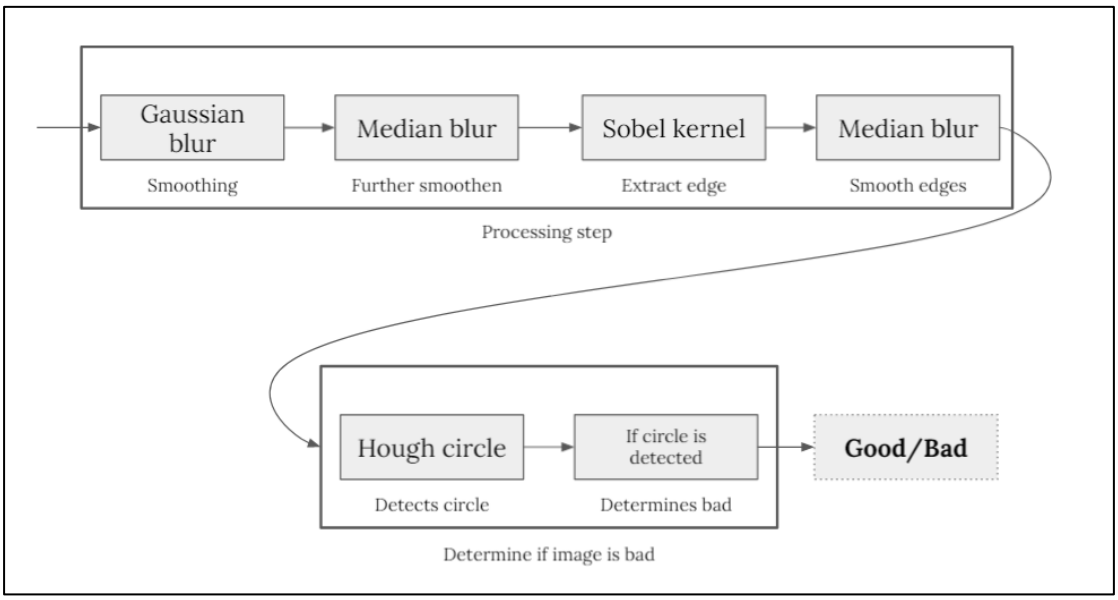


Figure 3.20 Overall process for bad bottom view bottle detection

3.4.4 Side view images

For side view, our main goal is to detect bottles and extract/crop the center bottle from the image. For detecting bottles, we used state of the art object detection algorithm Yolov5 [21]. The architecture of the model is shown in the **figure 3.21** below.

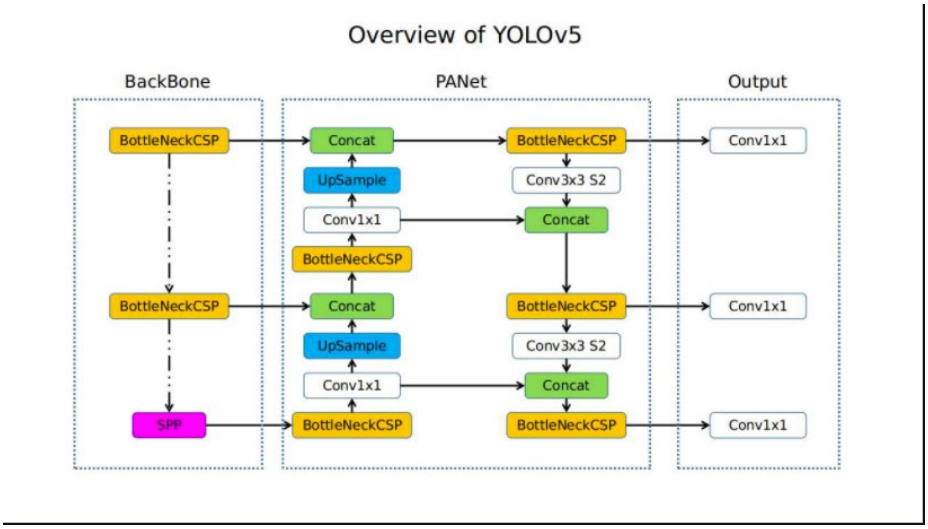


Figure 3.21 Architecture of Yolov5 [21]

As shown in the **figure 3.21** above, Yolov5 architecture is divided into three key features: BottleNeckCSP, BottleNeckCSP and SPP.

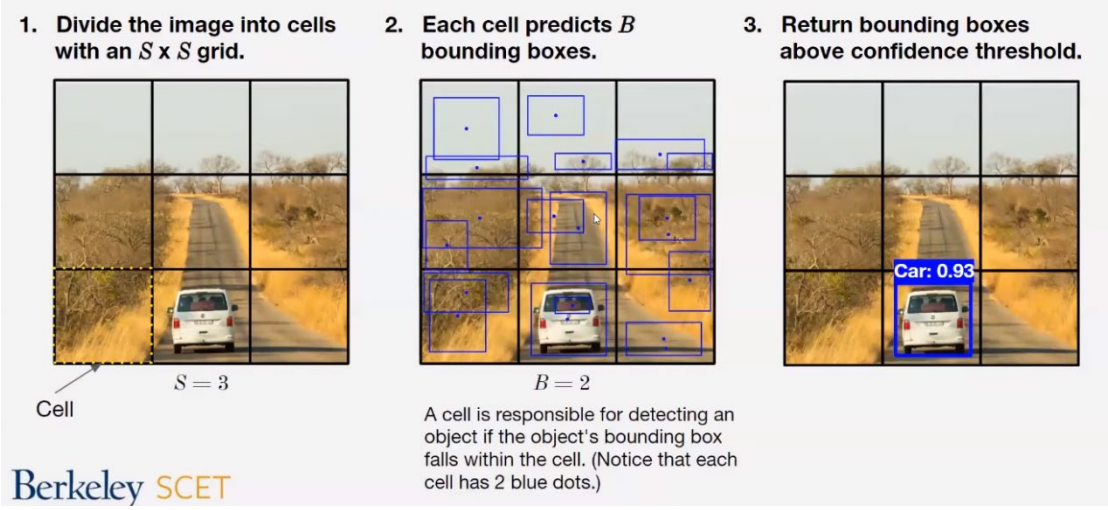


Figure 3.22 Overview for Yolov5 [22]

The above **figure 3.22** provides the general overview of Yolov5. Firstly, Image is divided into S x S Grid cell. For each individual grid cell, the algorithm predicts the boundary boxes. Then based on the confidence threshold most of the boxes are discarded and Non-Max Suppression is applied. So, the output is the boundary boxes with the confidence score [22].

3.5 Miscellaneous

3.5.1 Closest image

Image similarity is measured using the cosine similarity measure. In cosine similarity measure, the angle between the vectors is measured. If the measure outputs value close to 0, it implies the vectors are very similar to each other.

$$Similarity\ (A, B) = \frac{A \cdot B}{\|A\| * \|B\|}$$

In our application we utilize this similarity measure on the recently tagged image and its image feature. When an image is tagged, the similarity measure is applied on the rest of the un-tagged images in the cluster. Given the values, the cluster is re-ordered from closest (measure close to 0) to furthest (measure close to 1). The main idea behind this feature is that, if the features are extracted well, then this method can

hopefully group together similar tags. For instance: If a bottle with cigarette is tagged, then the next closest image should also have cigarettes in them.

3.5.2 Tag conflicts

Tags given by different labelers might not be consistent. Given the same set of images to different labelers, there might be certain images that have different tags. Our application allows users to select a dataset and find conflicts in the given dataset. When requested, the application updates the datasets directory from Google Drive with the latest tagged spreadsheet of all the users. Then all images that do not have conflicts are stored under the directory name after its tag. Image with conflicts is then displayed with information about the tags given to the image. This information can then be utilized to find the cause of the discrepancy.

3.5.3 Optimal cluster for created dataset

When creating the dataset, the application automatically clusters the directory using Kmeans. But instead of blindly choosing a cluster amount, the application tries to find the cluster amount with the best clustering score. Mean silhouette coefficient (more details in chapter 2.3.6) is used to measure the clustering performance. Higher the mean coefficient, better the clustering. The application goes through clusters from 2 to 50 and keeps track of the highest clustering score. The cluster amount with the highest score is then chosen as the number of clusters for Kmeans.

3.6 Chapter Summary

This chapter described the high-level requirements and design of a system that help labelers to classify bottles and reducing time for checking the bottle manually.

The design of labeling application and algorithm of detecting bad images is covered in further detail in Chapter 4 which describes the implementation of UI interface and method for detecting bad images.

CHAPTER 4
EXPERIMENTAL RESULT

4.1 Introduction

Chapters 3 and 4 described the design and implementation of bottle labeling system, a system that help the labelers to speeding up their process. In this chapter, we present a testing method and its results that show labeling application and algorithm of detecting bad images in different views works. The chapter is organized as follows: section 4.2 introduces system application and describes its UI interface and packaging of the application. Next, section 4.3 presents Application's UI/UX Evaluation.

The results of the testing and evaluation are summarized in section 4.4, before the chapter summary in section 4.5.

4.2 System Application

4.2.1 UI interface

The application divided in 3 tabs, which is cluster profile, configuration, and tag conflict. Each of tabs has different roles.

4.2.1.1 Cluster profile tab

The cluster profile tab allows users to tag the dataset. When no dataset is opened through the application, the cluster profile tab is blank with browse section visible.

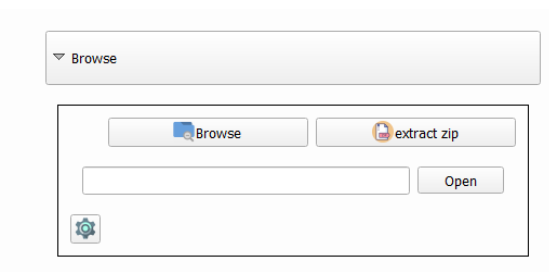


Figure 4.1 Browse section

The user can click the browse or extract zip button to open the dataset. This button will open a file browser where the user is required to specify the dataset directory as **figure 4.1** shown above. Once the appropriate directory is selected the user is prompted with 3 windows:

- Name prompt: Here the application request for user's name as **figure 4.2** shown below. This information will then be stored in the dataset.

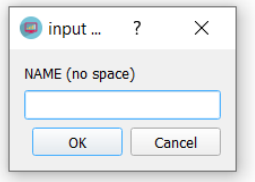


Figure 4.2 Name prompt

- Resume session: If the users have tagged the dataset before, this window is displayed to ask the users whether they would like to resume previous session as **figure 4.3** shown below.

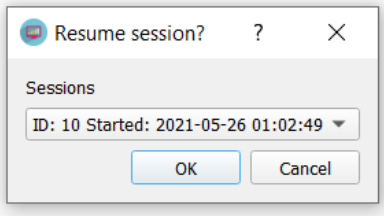


Figure 4.3 Resume session

- Focus tag: The final window asks whether the users would like to focus on any tag class (mentioned in chapter 3) as **figure 4.4** shown below. The application, for now, only allows the user to choose 1 tag at a time.

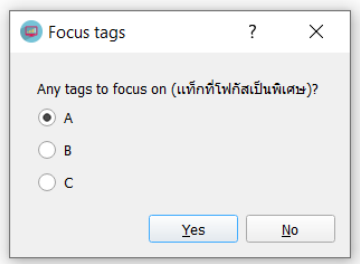


Figure 4.4 Focus tag

Once these windows are resolved, the application loads the dataset and displays the images with the predicted tag of the image, number of images tagged, number of images left to tag and UI to browser through the directory as **figure 4.5** shown below.

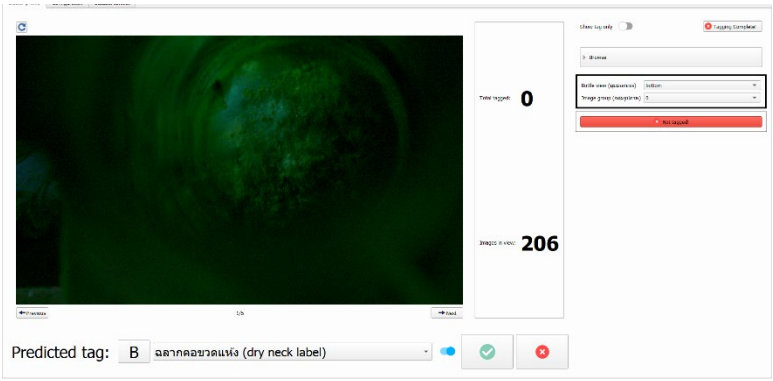


Figure 4.5 Cluster profile tab with predicted tag mode

In the predicted tag information UI, the users can choose to confirm or decline the prediction. The UI switch placed next to the confirm button allows users to choose between tagging the image as their subcategory or only the tag class as **figure 4.6** shown below. If the prediction is incorrect, the application displays the list of all tags, and the users can surf through them (either by scrolling through the list or searching the tags using the search input as **figure 4.7** shown below) to choose the correct tag. Using the search input the users can also create new tags.

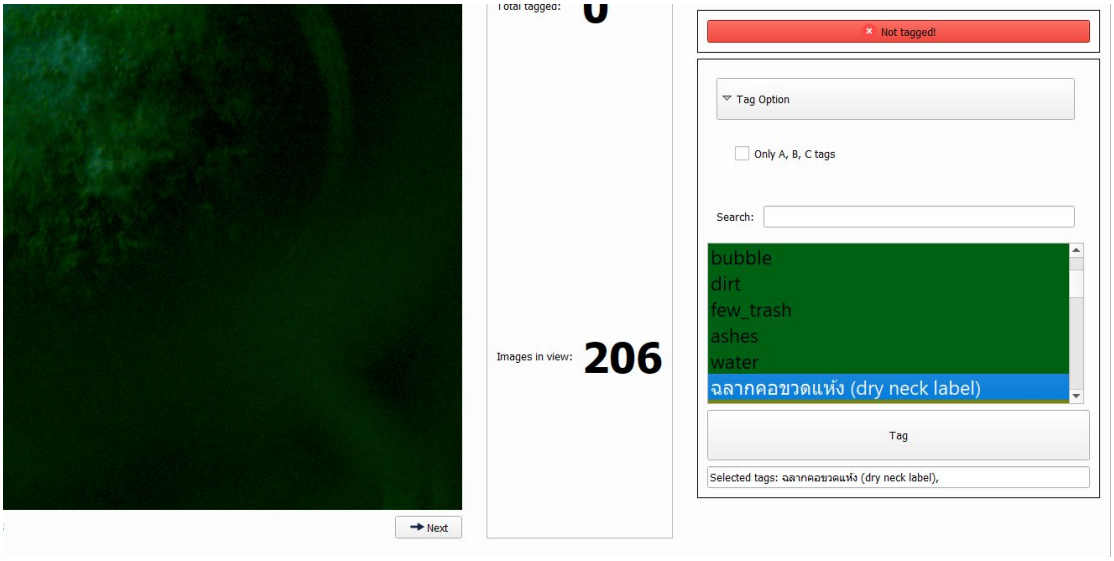


Figure 4.6 Cluster profile tab with all tags

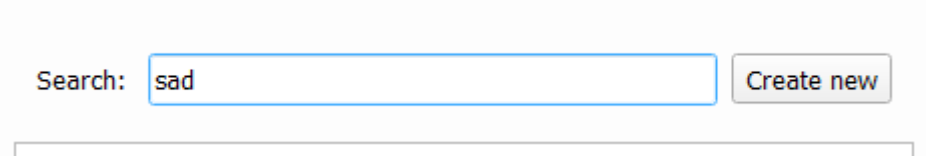


Figure 4.7 Searching bar in all tags

Using this tab, the experts can choose to re-cluster the directory. This option is available by clicking on the settings icon under the browse category. By clicking this button, a window pops up, showing available clustering algorithms and their parameters as **figure 4.8** shown below.

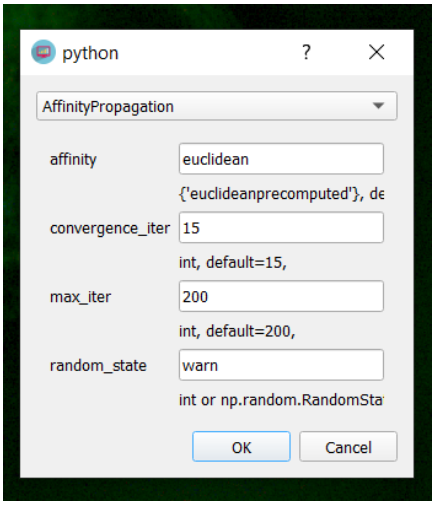


Figure 4.8 Clustering algorithms setting

4.2.1.2 Configuration tab

This tab allows users to convert a directory into an image directory. Essentially creating the local database, feature data, etcetera (mentioned in chapter 3). The application displays the allows users to choose the following configuration before the dataset is created:

- Feature extraction: A dropdown menu that displays list of pretrained model as **figure 4.9** shown below. The chosen model will be used to extract features from the images.

Figure 4.9 Feature extraction

- Tag list: The list of tags is stored on Google Drive. The UI displays these tags, directs users to the tag list, if they would like to change it and an update button that will update the tag list by downloading the tag list from Google Drive as **figure 4.10** shown below.

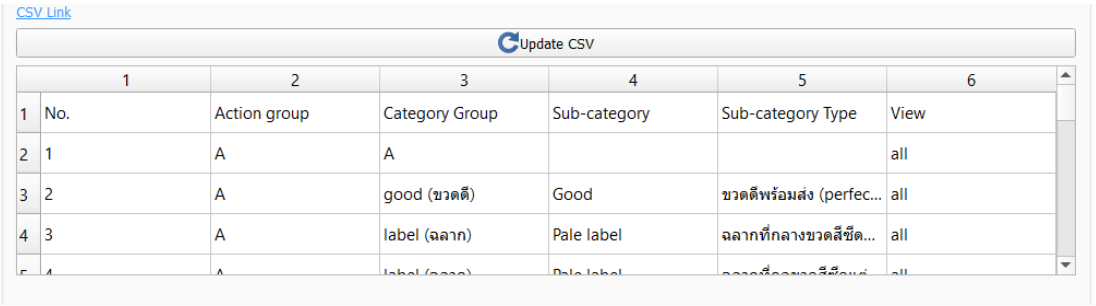


Figure 4.10 Tag list

- Tag configuration: Here the users can choose the saved parameter they would like to use for predicting the tags class of an image for each view as **figure 4.11** shown below.

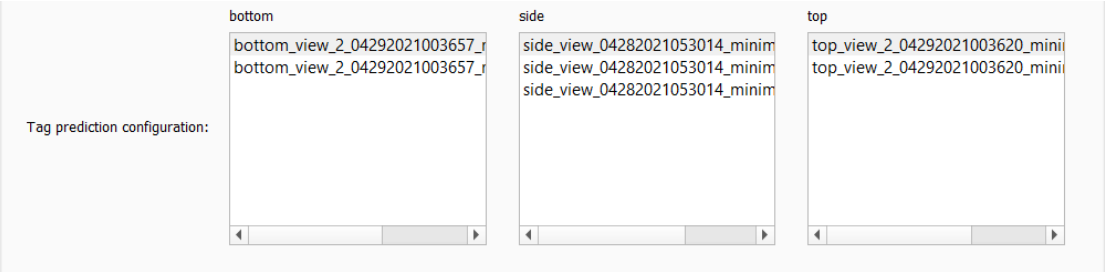


Figure 4.11 Tag configuration

Once all the configurations are chosen, the users can initiate the dataset creation process.

4.2.1.3 Tag conflict tab

The tab displays image conflicts (as described in chapter 3). Here, once the dataset is chosen, all the images with no conflicts are under the directory with the name of their tags. On the other hand, the images with conflicts are displayed in this tab as **figure 4.12** shown below.

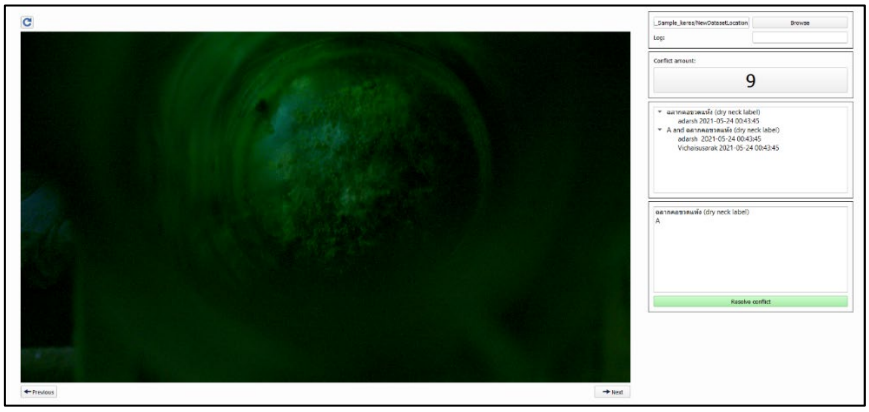


Figure 4.12 Tag conflict tab

Here, certain UI elements are like the tagging tab. The following elements are added in this tab:

- An UI element displaying the number of images with conflicts as **figure 4.13** shown below.

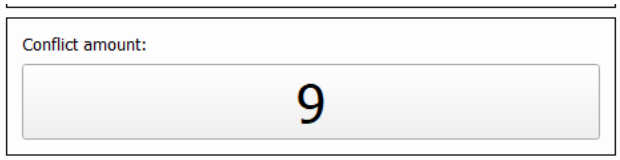


Figure 4.13 Displaying conflicting images number

- Showing list of tags different users tagged the given image as. Here the main list contains the tag. And the list under each tag contains name of the users who gave that tag as **figure 4.14** shown below.

Figure 4.14 Show the different users tag on image

- And the final element is the resolve conflict section. This section is not necessarily used. The main purpose of conflict tab is to display the set of images with conflicting tags and the users who tagged them. But if after discussion, a certain set of tags is agreed upon to be the true tag of the image, the users can choose these tags and resolve the conflict as **figure 4.15** shown below. Which will then move the images to a directory named after the set of tags.

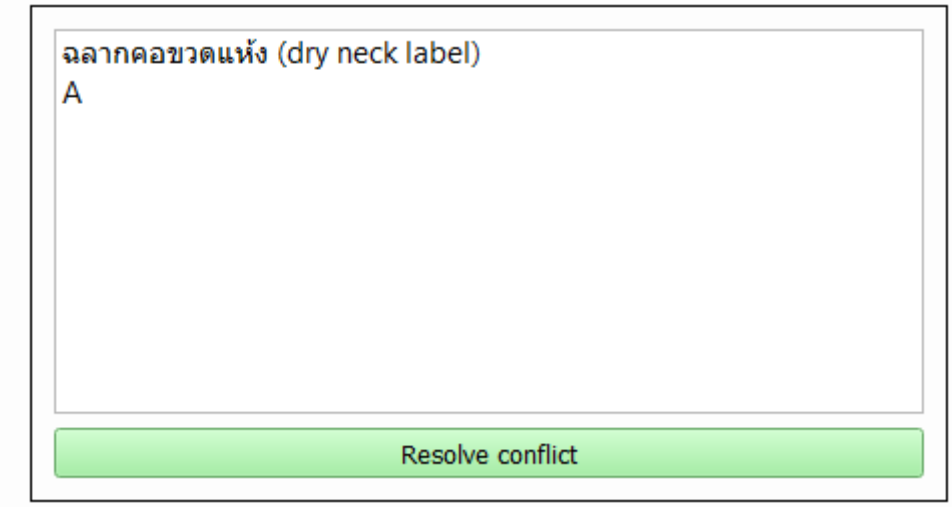


Figure 4.15 Resolve conflict

4.2.2 Packaging Application

The packaging application can be separated in to 2 parts:

- Converting Python file to an EXE file
- Create an application installer

For converting Python file to an EXE file, it is simply by just going to directory that ready for converting and open command prompt, type the command line of Nuitka, which is python -m nuitka --mingw64 --windows-disable-console --windows-icon-from-ico= [**YOUR DIRECTORY OF THE APPLICATION ICON] [PYTHON FILE NAME]** as **figure 4.16** shown below, and Nuitka will start to covert selected python file to an EXE file as **figure 4.17** shown below. After that you will receive an EXE file that name exact the same as your Python file name as **figure 4.18** shown below. In addition, An EXE file can be opened with any computer that run Window 10 and do not require any Python and its library to run it.

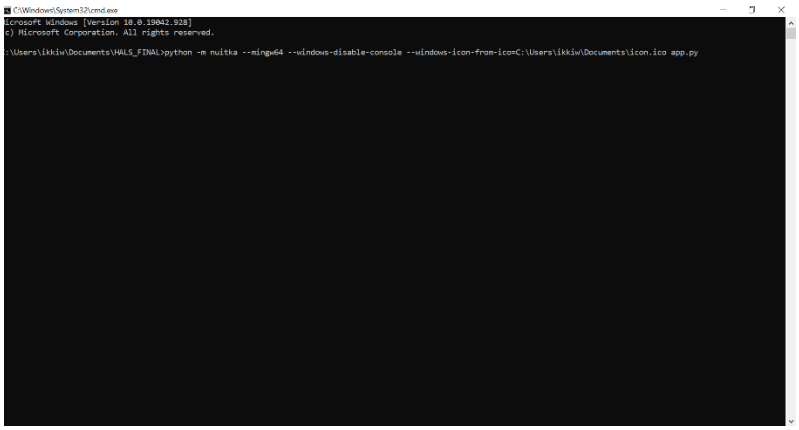


Figure 4.16 Python conversion Nuitka command line

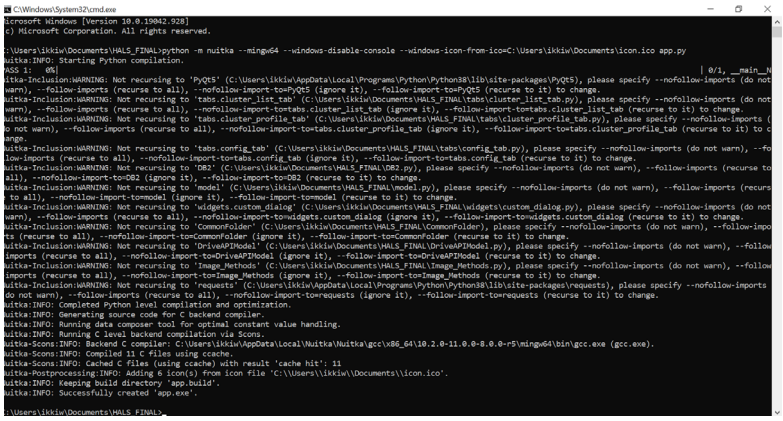


Figure 4.17 Process of converting Python to EXE

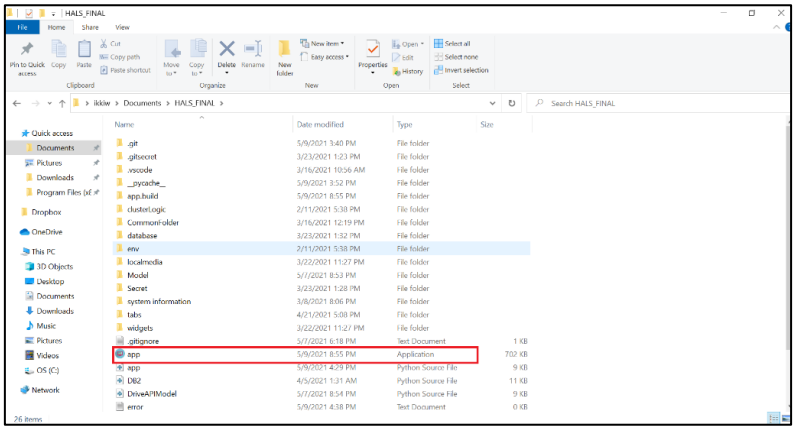


Figure 4.18 EXE in file explorer

For Installer and updater, there will be files and folders selection on Advance Installer that want to make the Installer as **figure 4.19** shown below and put the sharable link of cloud storage such as Dropbox or Google Drive to make an updater as **figure 4.20** shown below in this case we put the Dropbox link. After that we build the Installer with Updater inside of it as figure 4.21 shown below then Installer file will be shown in file explorer as **figure 4.22** shown below. Consequently, when we install the application by installer, there will be Launch the application checkbox for launching application after the installation is done as **figure 4.23** shown below. Finally, there will be a folder in your selected installation folder, which included all necessary dependencies for application and updater as **figure 4.24** shown below. In addition,

installer will automatically create the shortcuts in both start menu and desktop for better usability for the users as **figure 4.25** shown below.

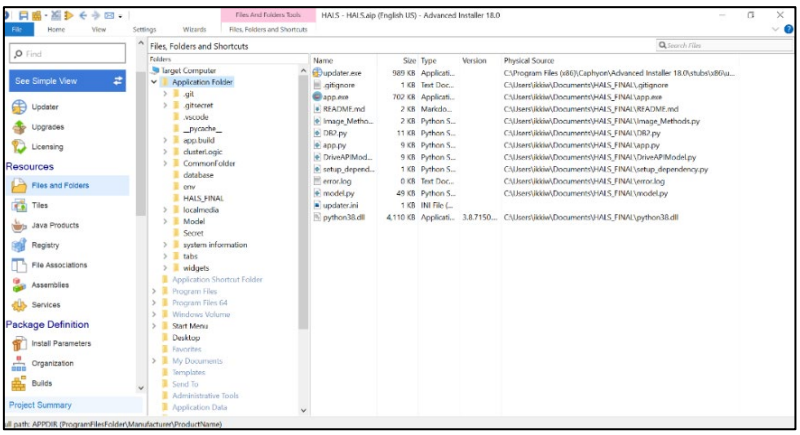


Figure 4.19 Target file selection for building installer

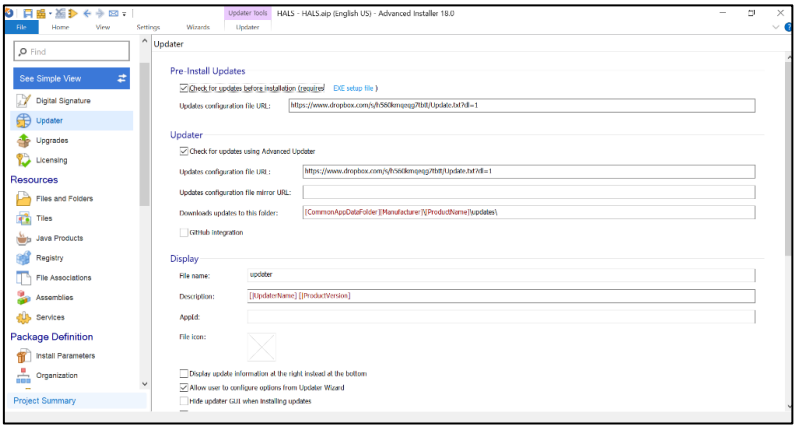


Figure 4.20 Pre-install updates and updater configuration

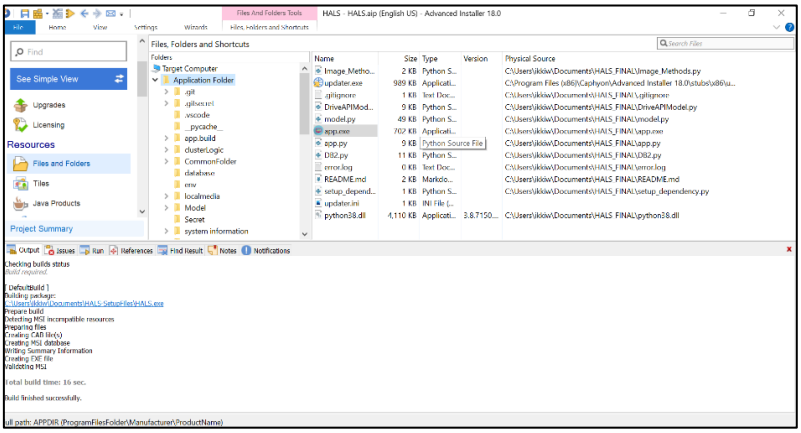


Figure 4.21 Building Installer process

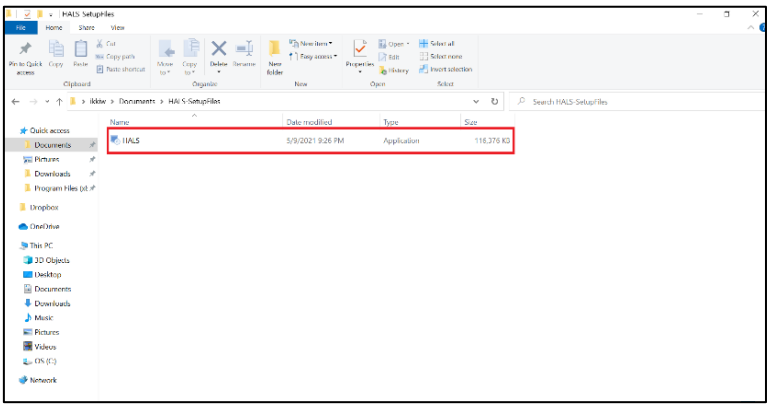


Figure 4.22 Installer in file explorer

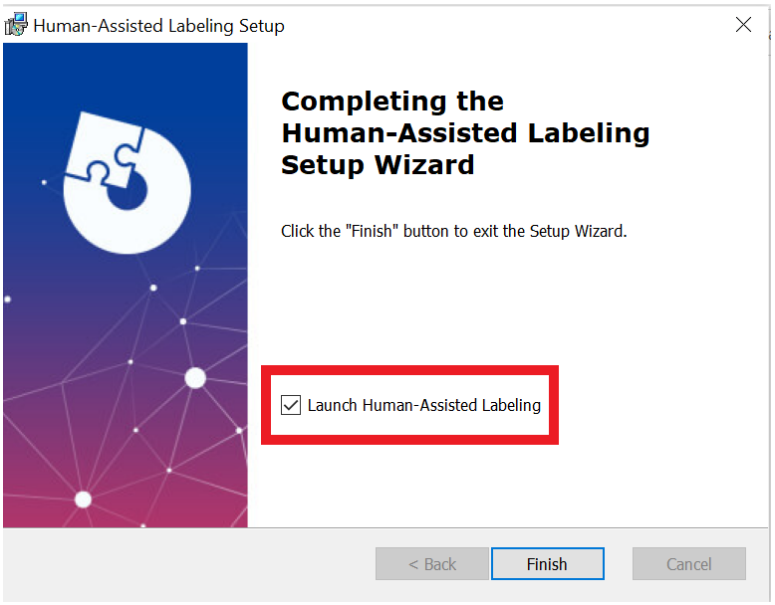


Figure 4.23 Launch application after installation is finished

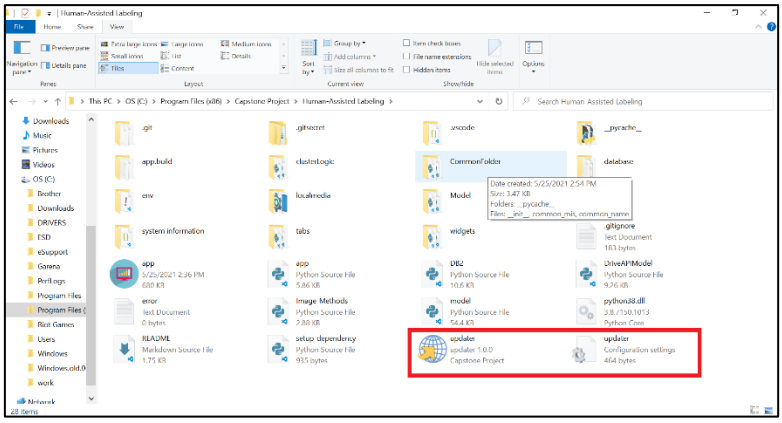


Figure 4.24 Updater after installation is finished

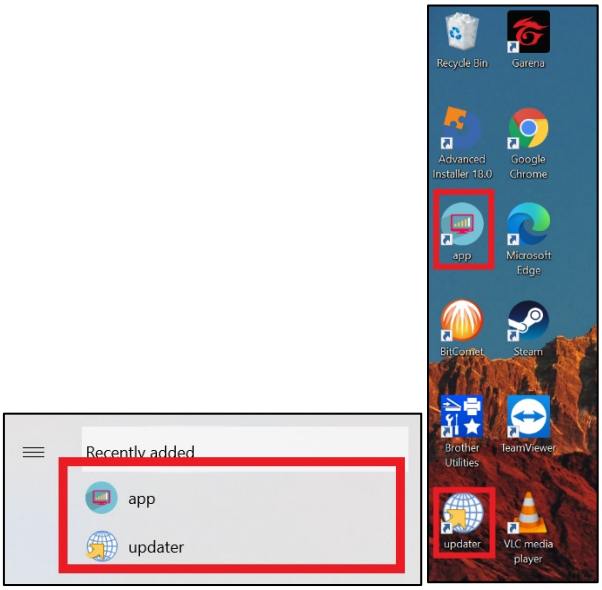


Figure 4.25 Shortcuts on start menu and desktop created after installation is finished

4.2.3 System detection of bad images algorithm

For top and bottom view, the goal of our algorithm is to have very high precision rate.

$$Precision\ rate = \frac{TP}{TP + FP}$$

Basically, if the images that are detected as bad, are bad, our algorithm can be deployed in our application. The main reason behind this is to make sure good images are not being discarded.

4.2.3.1 Detecting Top views

The algorithm for top view image, discussed in chapter 3, was tested a directory with total of 296,702 images. The result of the experiment is shown below.

Table 4.1: Result of top views

DIRECTORY	TOTAL IMAGES	bad image [predicted]	% bad image [predicted]	bad image [actual]	% bad image [actual]
F:\images\top_view	296,702	197	0.066	196	99.49238579

As we can see from the data above, out of all the images in the dataset, the algorithm detected 197 images as bad. By surfing through the 197 images, only 1

image did not meet the “bad” image criteria. The other 196 images either did not have any bottle, did not contain the entirety of the top view or the image was taken from an unusual angle.

Figure 4.26 Blurred images

The only good image was not perfect either. Certain potion of the top view was heavily blurred as **figure 4.26** shown above. But since the entirety of the top view was present and the properties of the top view could be interpreted (deformed, etcetera), we classified this image as good as **figure 4.27** shown below.

Figure 4.27 Image that classified as good

Since our algorithm's goal is to maximize the precision rate, and the top view algorithm has a precision rate of 99 %, we consider this algorithm satisfactory and implemented in our application. The algorithm also worked on a large dataset, with images taken from May to November of 2020. Further proving the robustness of the algorithm as **figure 4.28** shown below.

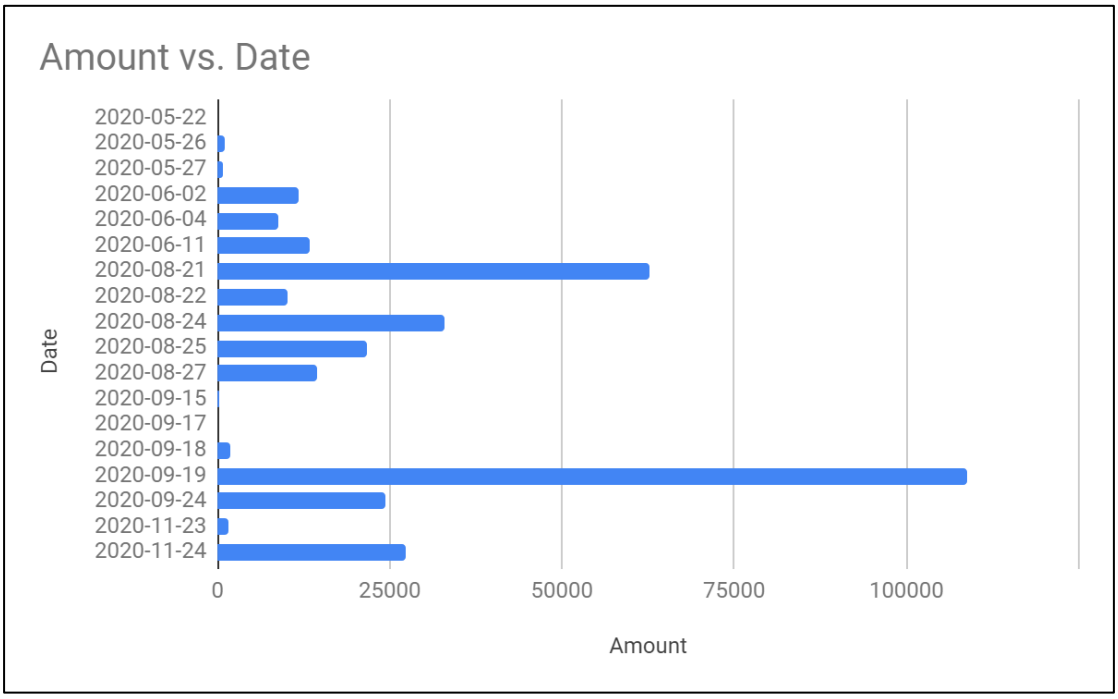


Figure 4.28 Amount of data used in date

4.2.3.2 Detecting Bottom views

In chapter 3, we mentioned a more involved edge detection method for bottom view compared to top view image. Initially, we utilized the simple edge detection and tested the performance of the algorithm on the dataset of size of 341,752 images. The stat of the experiment is shown below.

Table 4.2: Result of bottom views

DIRECTORY	Total images	bad image [predicted]	% bad image [predicted]	bad image	Good image	Unsure
F:\images\bottom_view	341,752	3,457	1.012	1,366	156	1,937

As we can see, 3,457 images were detected as bad. Out of this, 1,366 were bad and rest were either good or lied in the gray area. Images that lie in the gray area, are images that do not have the entirety of the bottom view in the image but does contain majority of it. It is hard to classify whether these images should be considered good or bad, since although the entire view is not present, enough is present to interpret the property of the bottle. Some of the images, in the unsure category, has the entire view, but the image has lighting issues as **figure 4.29** shown below.

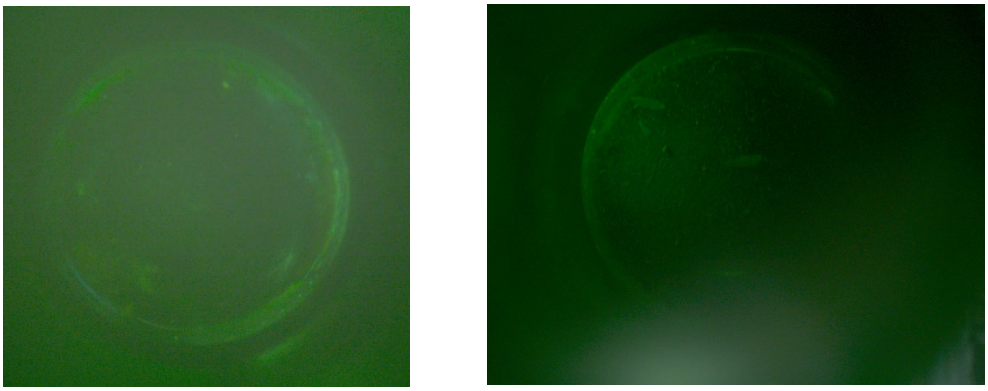


Figure 4.29 Lighting issues

Due to the high amount of unsure and good images using the simple edge extraction method, we modified our extractor (mentioned in chapter 3) and tested it on the same dataset while varying the param2 parameter of Hough circle. Param2 is the parameter that thresholds the acceptance of a given circle. If param2 is set high, only perfect circles are accepted. Whereas if param2 is set very low, even semi or deformed circles will be accepted. In the previous experiment, using the simple edge extractor, the param2 value was set at 60. The table below shows the performance of detecting bad image with modified edge extractor on the same dataset.

Table 4.3: Performance of param2

Param2	Amount detected	Actual Bad	Unsure	Precision Rate
60	1,236	1,136	100	0.92

50	814	804	10	0.99
40	323	323	0	1

As we can see, when the param2 is set at 60, fewer images are detected as bad (compared to the method that utilized the simple edge extractor). On top of this, there exists no true false positives. But again, there were few images that lied in the gray area. We could have classified these images as good or bad by asking opinions from experts, but in this case, we will just assume the worst case and consider the unsure images as false positive. Given this assumption, the precision rate when param2 is set at 60 is 92%. By lowering param2, we found that fewer images were being detected as bad, as well as the precision rate increased. Where param2 of 50 had very high precision rate, as well as more bad images were detected compared to a lower param2 value (40). Given this precision rate meets our criteria, we integrated this algorithm in the application.

We also tested the difference in processing time between using simple (top view) and modified (bottom view) edge extraction method. In this experiment, we sampled 500, 1,000, 10,000 and 50,000 images and measured the amount of time both methods took to extract the edges.

Table 4.4: Comparison between bottom view and top view

Image amount	**Old method (top view)**	Average/image	**New method (bottom view)**	Average/image	% increase from old
500	265.55	0.53	269.69	0.54	1.56%
1,000	435.75	0.44	721.311	0.72	65.53%
10,000	5,469.39	0.55	6,172.93	0.62	12.86%
50,000	34,774.07	0.70	38,692.75	0.77	11.27%

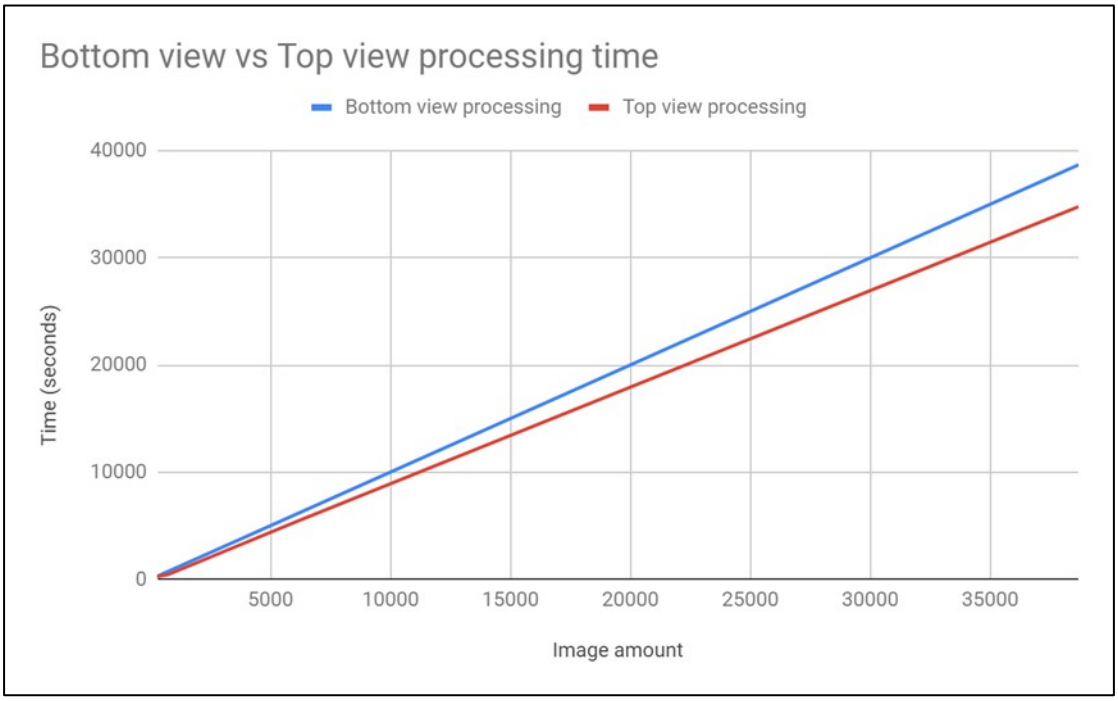


Figure 4.30 Time spending comparison between bottom view and top view

As we can see, on average the bottom view processing time is 22.81% slower than the Top view edge extraction method. On average, the Bottom view takes 11 milliseconds more time to process an image compared to the Top view as **figure 4.30** shown above. This increase in processing time is expected due to more steps involved in the modified edge extractor.

4.2.3.3 Detecting Side views

The algorithm used to detect side view images is discussed in Chapter 3. In order to detect side view images, we need to prepare a training and validation set to train the model. For labeling the images to create training and validation set, we use an image annotation tool called LabelImg (as we mentioned in chapter 2). This is a free software tool that helps us to create labels based on Yolo format.

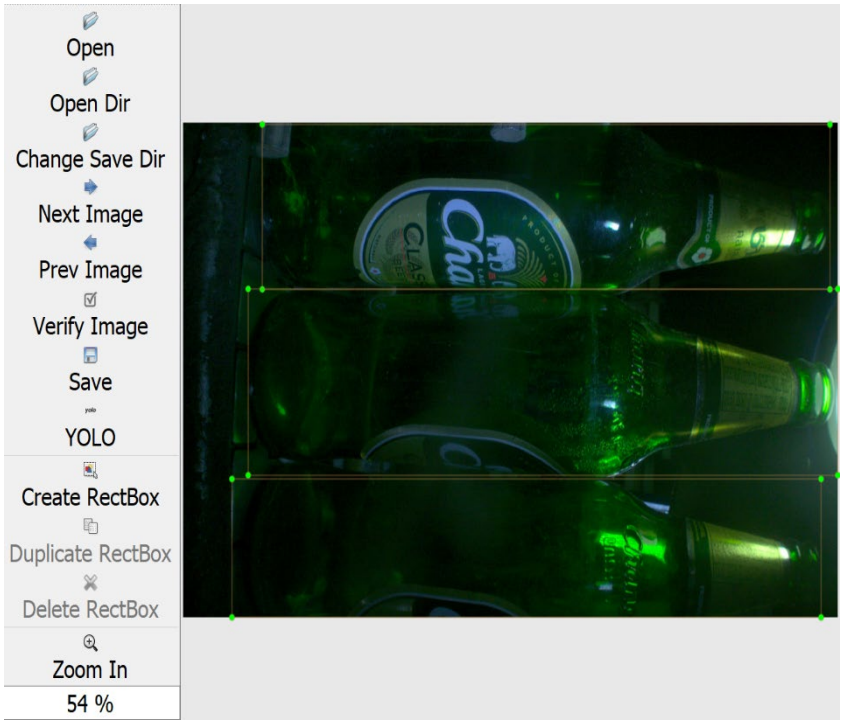


Figure 4.31 Interface of LabelImg

The above **figure 4.31** shows the interface of LabelImg. In the labelling process, we carefully label the images by creating a rectangular boundary box.

Once the labelling process is done, we move to training the algorithm. We conducted multiple experiments to determine the best batch size, confidence threshold, epoch size for the model. We observed batch size of 25, confidence threshold of 0.5 and epoch size of 600 to best fit our model. In the process of conducting experiments, we visualize different performance metrics such as loss function, precision, mAP and recall.

Figure 4.32 Training Loss and Box Loss

In the above **figure 4.32**, we visualize two experiments i.e., experiment 2 and experiment 7. We observe training loss to check if our model is performing well or not. Since the graph is steadily decreasing, we can say that the model is performing well and our chosen paraments like batch size and epoch size are good.

Figure 4.33 Validation Loss and Box Loss

Similarly, we visualize validation loss to observe if our model is overfitting or not. Since the validation loss is also constantly decreasing, we can say that our model is performing well and is not overfitting as **figure 4.33** shown above.

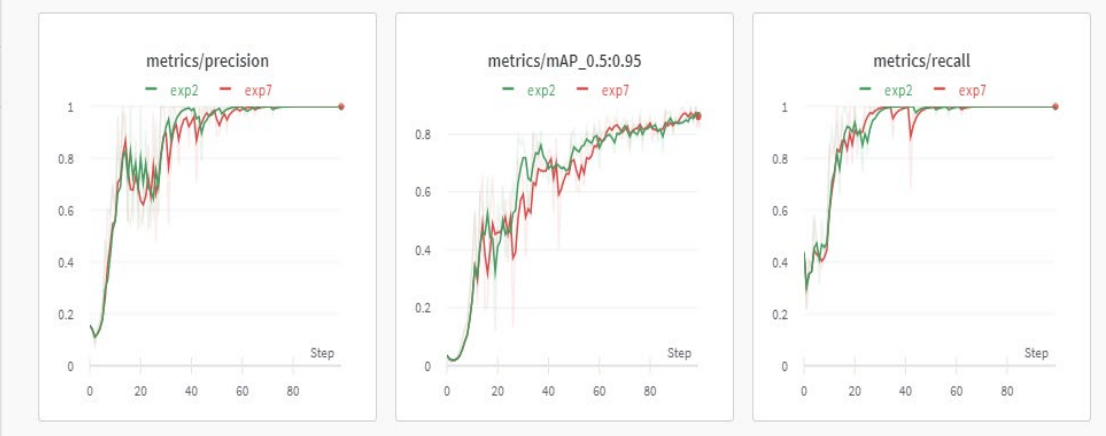


Figure 4.34 Precision, mAP and Recall for exp2 and exp7.

Precision refers to the identification of only relevant object. Precision is the percentage of true positive and is given by:

$$Precision = \frac{TP}{TP + FP}$$

Similarly, Recall is the percentage of true positive detected and is given by:

$$Recall = \frac{TP}{TP + FN}$$

For object detection algorithm, mAP is a fully encompassing metrics that gives us true look at the performance of our model [23]. Therefore, we mainly focus on mAP@[.5:.95]. This refers to Average Precision over different IOU thresholds from 0.5 to 0.95 with step of 0.05. In our experiment, mAP is constantly increasing and is greater than 0.8. This proves that our model is performing good as **figure 4.34** shown above.

Table 4.5: Time taken for different batch sizes

Image	Batch Size	Time Taken(sec)

Left view	30,000	3,609.5
Left view	30,000	3,020.2
Left view	30,000	3,540.0
Left view	30,000	3,202.0
Left view	30,000	3,530.0
Left view	20,000	2,080.4
Right view	30,000	3,760.5
Right view	30,000	3,426.7
Right view	30,000	3,782.4
Right view	30,000	3,320.0
Right view	30,000	3,816.5
Right view	30,000	3,200.5
Right View	10,000	1,252.7

Once we complete the training process, we run the inference on more than 400,000 side view images. The result and time taken for the images are shown above:

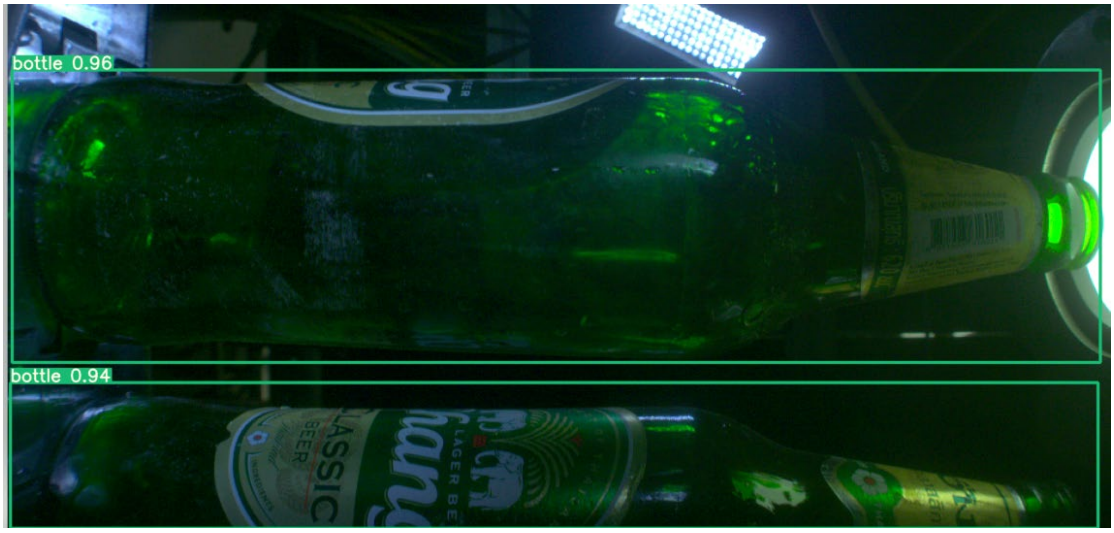


Figure 4.35 Inference result

As we can see from the image above, the detection of bottles is good and with the confidence score of 0.96 and 0.94 as **figure 4.35** shown above.

4.3 Labeling application's UI/UX Evaluation

4.3.1 Labeling application survey result

We have doing labeling application Survey on 17 March 2021 by inviting Thaibev staff to be using our application to tagging bottle images until they have ended tagging images session in dataset and after that given us the evaluation score and feedback to our improvement because we want to gather information for the application improvement in real scenario of real users using our application in real work. However, according to Covid-19 situation, we had plan to do 2 surveys on the UI/UX evaluation by 1st will be held on March 2021 and 2nd will be in May 2021, but the Thaibev staff was not available to came to CMKL to tagging the images, so we will have only 1 survey.

4.3.1.1 Evaluation tools and criteria

Project evaluation was created by creating a questionnaire to evaluate satisfaction rate of real labelers about usability and design of the application by having score criteria as **Table 4.6** shown below.

Table 4.6: Evaluation level

Evaluation Level	Value	Satisfaction Level
Very Satisfied	5	Very satisfied with the application
Satisfied	4	Satisfied with the application
Fair	3	Moderately satisfied with the application
Dissatisfied	2	Not satisfied with the application
Very Dissatisfied	1	Not very satisfied with the application

4.3.1.2 Evaluation results

From the **Table 4.7** shown below, we can see that on March result is quite satisfy to the labelers for using the application with overall score is 32/40 (80%) and got least score as 3, which is Fair for 3 questions as **Table 4.7** shown below. However, according to this survey we still improve more of application UI/UX after March survey is done to ensure that we can deliver the application that really help them to label bottle faster and easier and minimize issues that might interrupted user's usability in the application. For example, we have changed image suggestion layout and its function to be automatically re-order cluster and find closet image function instead to reduce complexity of the application. **Figure 4.36 - 4.42** shows the actual results of questionares score.

Table 4.7: Question and score

Question	Score
Does images suggestion placement and layout appropriate?	3
Is suggested image easy to use (Function)	3
Is screen layout used in the Tagging tab appropriate?	3
Is the arrangement of tags appropriate?	4
Is tags selection simply to use?	5
Is font size used in the Tagging tab appropriate?	4
Is font style used in the Tagging tab appropriate?	5
Is searching tag easy to use?	5
Total	32/40

4.4 Testing and evaluation Summary

An evaluation of labeling application was then presented in section 4.3. Section 4.2 revisited the requirements described in Chapter 3 and identified that labeling application (section 4.2.1), packaging application (section 4.2.2) and method for detecting bad images (section 4.2.3) their performance and usability.

4.5 Chapter Summary

In conclusion, this chapter introduced demonstration labeling application, which is consist of different tabs for different usability as mentioned above. And performance of method for detecting bad images in different views (top, bottom, side). Consequently, application's UI/UX evaluation survey of Thaibev staff. For the next chapter, we will be talking about the conclusion of bottle labeling system.

CHAPTER 5
CONCLUSION

5.1 Introduction

In this Chapter, we first summarize the work described in this report (section 5.2). Then we draw a number of conclusions about key parts of the work undertaken in section 5.3, and finally in section 5.3.1 we discuss about our work in general, which is separated by labeling application and method (section 5.3.1.1) to find bad images (section 5.3.1.2).

5.2 Summary

This is a summary of each chapter intro and summary:

Chapter 1 introduced overall of our project description and what is we expected on this project.

Chapter 2 reviewed the literature in bottle labeling system. Background of project was introduced, and tool used in project described. The potential for related work was highlighted.

Chapter 3 describes the design of bottle labeling system. We separate functions of application that support the requirements were then described in more detail, including use case diagram of different roles and sequence diagram of functionality of application. Also, describing the method of detecting bad bottle images with different views, which is top, bottom, and side view.

Chapter 4 described the implementation labeling application, method of detecting bad bottle image, and survey result of labeling application in March.

Chapter 5 presented conclusion for bottle labeling system and discussion about bottle labeling system.

5.3 Conclusions

In conclusion, the goals of this project were to help labelers to classify bottles for recycling faster. We chose to focus on making application to help labelers to tag images with automate features to help them to classify bottle faster. Also, creating method of detecting bad images with different views (bottom, side, and top) to precisely detect bad images in dataset.

We then designed and implemented a system that could:

- Provided an application interface for the users to create dataset, browse images dataset and tag each image, which provided features that help support for tag image easier and faster such as feature extraction, predicted image, resume session, switch cluster and re-order images, etc. as we mentioned on chapter 3 above.
- After the tagging process is done, the application can store the data in both local (SQLite) and cloud storage (Google Drive) to make users to track performance (Model configuration) and validate (Image tag data).
- Utilize method for detecting bad images to help the users to detect bad images in dataset from different types of views.
- Package application for the users to reduce complexity of usability in application.

These combined capabilities of developing application in Python, improving its UI/UX, and testing result from method of detection bad images, and packaging application.

REFERENCES

[1] N. R. Council, *Glass Recovery Hierarchy and Environmental Benefits Snapshot*, Oct.15, 2019. Accessed on: May 24, 2021. [Online]. Available: https://nerc.org/documents/Glass/glass_hierarchy_oct_15_2019.pdf.

[2] PyQt, *Qt for Python*, n.d. Accessed on: May 24, 2021 [Online]. Available: https://doc.qt.io/qtforpython/.

[3] SQLite, *What is SQLite?*, n.d. Accessed on: May 24, 2021 [Online]. Available: https://www.sqlite.org/index.html.

[4] Google Developers, *Python Quickstart*, May. 4, 2021. Accessed on May. 24, 2021. [Online]. Available: https://developers.google.com/docs/api/quickstart/python

[5] Wiki, *Nuitka*, May 7, 2021. Accessed on May. 24, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Nuitka

[6] "*Advance Installer"*, n.d. Accessed on May. 24, 2021. [Online]. Available: https://www.advancedinstaller.com/

[7] Wiki, *Gaussian Blur*, May 8, 2021. Accessed on: May 24, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_blur#:~:text=In%20image%20processing%2C%20a%2 0Gaussian,image%20noise%20and%20reduce%20detail..

[8] OpenCV, *Hough Circle Transform*, May 24 ,2021 Accessed on: May 24, 2021. [Online]. Available: https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

[9] IPAC, *Line Detection by Hough*, Apr. 20, 2009. Accessed on: May 24, 2021. [Online]. Available: transformationhttp://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf.

[10] "*Median Blur*", n.d. Accessed on May. 24, 2021. [Online]. Available: https://docs.gimp.org/2.10/en/gimp-filter-median-blur.html

[11] Ultralytics, *YOLOV5*, n.d. Accessed on May. 24, 2021. [Online]. Available: https://github.com/ultralytics/yolov5

[12] Satyam Kumar, *Silhouette Method — Better than Elbow Method to find Optimal Clusters*, Oct. 19, 2020. Accessed on: May 25, 2021. [Online]. Available:

https://towardsdatascience.com/silhouette-method-better-than-elbow-method-to-find-optimal-clusters-378d62ff6891
[13] Selva Prabhakaran, *Cosine Similarity – Understanding the math and how it works (with python codes),* n.d. Accessed on: May 25, 2021. [Online]. Available: https://www.machinelearningplus.com/nlp/cosine-similarity/
[14] tzutalin, *LabelImg*, n.d. Accessed on: May 25, 2021. [Online]. Available: https://github.com/tzutalin/labelImg
[15] Joris Guérin, *CNN features are also great at unsupervised classification,* Jul. 6, 2017. [Online]. Available: https://arxiv.org/abs/1707.01700
[16] Keras, *ResNet and ResNetV2*, n.d. Accessed on May. 24, 2021. [Online]. Available: https://keras.io/api/applications/resnet/
[17] ScikitLearn, *2.3.2. K-means*, n.d. Accessed on May. 24, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html.
[18] Parnmet Daengphruan, Orathai Sangpetch and Akkarit Sangpetch, DASSL: Dynamic, AI-assisted, Scalable System for Labelling Used Bottle Images. n.d.
[19] Scipy, *numpy.memmap*, Jul. 21, 2020. Accessed on May. 24, 2021. [Online]. Available: https://docs.scipy.org/doc/numpy-1.4.x/reference/generated/numpy.memmap.html
[20] OpenCV, *Hough Circle Transform*, May. 25, 2021. Accessed on May. 24, 2021. [Online]. Available: https://docs.opencv.org/master/da/d53/tutorial_py_houghcircles.html
[21] seekFire, *Overview of model structure about YOLOv5*, Jul. 3, 2020. Accessed on May. 24, 2021. [Online]. Available: https://github.com/ultralytics/yolov5/issues/280
[22] SCET Berkeley, *YOLO Object Detection (Part 1)*, Aug. 10, 2020. Accessed on May. 24, 2021. [Video file]. Available: https://youtu.be/2hAiJe8ITsE
[23] blueskywwc, *How to improve Precision?,* Nov. 10, 2020. Accessed on May. 24, 2021. [Online]. Available: https://github.com/ultralytics/yolov5/issues/1335