Final Report

The Son Ngo – 103827804 | Phong Tran – 104334842 | Asher Nagel – 103592160 | Thanh Dang - 103824041| Quoc Co Luc - 103830572

Studio: 1-5 Group No. 2 Theme 1

1. Introduction

1.1 Background and Motivation

Our motivation for this project arises from the increasing need for automated solutions in urban waste management to keep growing cities clean and environmentally sustainable. In particular, illegal dumping and improper waste disposal are common problems in urban areas, impacting public health, safety, and aesthetics. Recognising this, our project aims to leverage AI to identify and classify roadside rubbish using image data, thereby offering a smart city solution that could assist in more efficient waste management.

This AI model will primarily benefit local governments, waste management agencies, and environmental monitoring teams responsible for maintaining public spaces. These users will be able to detect rubbish quickly, distinguish between different types of waste, and use this information to prioritise areas that need immediate attention. By providing insights into waste patterns and disposal trends, our model can also help city councils make data-driven decisions that support sustainability efforts.

1.2 Project Objectives

The primary objective of this project is to develop a computer vision application capable of detecting rubbish in roadside images and classifying it based on its type. Key questions we aim to answer include: Can we reliably identify rubbish in diverse roadside conditions? And can the model classify different rubbish types to aid in targeted waste management?

Through this project, we aim to deepen our understanding of computer vision applications, particularly object detection and classification within the context of smart cities. Our AI models present a practical tool for optimising urban waste management and enhancing environmental monitoring. This project also benefits us as we gain hands-on experience in handling real-world data, annotation, model training, and performance evaluation, all of which are essential skills in AI development.

1.3 Summary of Outcomes

Our final AI model demonstrated promising performance in detecting and classifying roadside rubbish. We compared the effectiveness of two models: YOLO and U-Net. The results indicated that YOLO outperformed U-Net in detecting rubbish objects, with better precision and recall rates. U-Net, while effective in some scenarios, struggled to accurately identify and classify the diverse types of waste present in the images.

The YOLO model was capable of identifying rubbish and differentiating between various types of waste objects, showing reliability across different lighting conditions and angles. These outcomes highlight the potential of utilising AI in urban waste management and lay a solid foundation for future research in automated environmental monitoring.

2. Dataset

2.1 Data Source

2.1.1 Overview dataset

This dataset is provided in the theme 1 folder inside the Design Project link, our team choose dataset based on rubbish and non-rubbish in the roadside photos at city council. Our work is to detect and classify the rubbish and familiar items inside that.

Statistics in 1000 images

- Rubbish: 806
- Non-rubbish: 194
- Image format: jpg

2.1.2 Limited label list:

"container", "waste-paper", "plant", "transportation", "kitchenware", "rubbish bag", "chair", "wood", "electronics good", "sofa", "scrap metal", "carton", "bag", "tarpaulin", "accessory", "rubble", "table", "board", "mattress", "beverage", "tyre", "nylon", "rack", "styrofoam", "clothes", "toy", "furniture", "trolley", "carpet", "rubbish".

2.1.3 Categories:

- Rubbish: The dataset contains the rubbish where it located at the roadside, where it might be various items from the most familiar and visible to unfamiliar and blurred. Inside the rubbish contains one or more items based on the unified list that team has given in the

2.1.2 section.

- Non-rubbish: When labelling the image, we have seen more than hundreds of images are non-rubbish (as seen on the **Statistics in 1000 images** in **2.1.1 section**). They are non-visible rubbish as we tried to find multiple times.

2.2 Data Processing

2.2.1 Data organisation:

Allocation: Every team member must label 200 images, and check if there is any rubbish with its inside or non-rubbish, therefore put the non-rubbish into another folder.

2.2.2 Folder structure

Root: main folder for the team

Team needs to label images below:

- \`phong\`, \`co\`, \`son\`, \`asher\`, \`thanh\`: containing 100 images
- \`phong2\`, \`co2\`, \`son2\`, \`asher2\`, \`thanh2\`: containing 100 images

After that, all rubbish images with label and non-rubbish images are put into the \`processed_data\` folder, inside that contains:

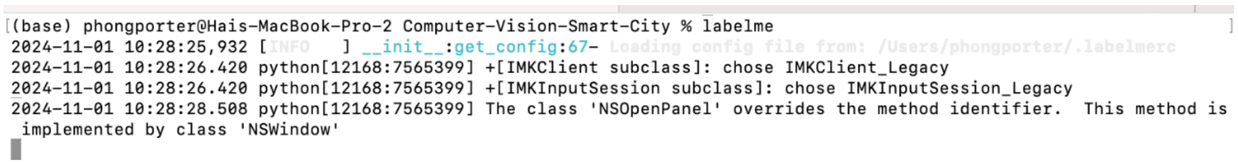
- \`phong\`, \`co\`, \`son\`, \`asher\`, \`thanh\`: each of the folder contains both \`rubbish\` and \`non-rubbish\` sub-folder
    - \`rubbish\`: Images with labels in JSON format
    - \`non-rubbish\`: images only

Application used for labelling data: *labelme*.

File name conventions: the image name is same as the label name.

2.2.3 Label processing step-by-step:

Step 1: Open labelme by installing its dependencies on terminal with virtual environment, then enter labelme to open the application



```
(base) phongporter@Hais-MacBook-Pro-2 Computer-Vision-Smart-City % labelme
2024-11-01 10:28:25,932 [INFO    ] __init__:get_config:67- Loading config file from: /Users/phongporter/.labelmerc
2024-11-01 10:28:26.420 python[12168:7565399] +[IMKClient subclass]: chose IMKClient_Legacy
2024-11-01 10:28:26.420 python[12168:7565399] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
2024-11-01 10:28:28.508 python[12168:7565399] The class 'NSOpenPanel' overrides the method identifier.  This method is
 implemented by class 'NSWindow'
```

Figure 0.1: Terminal MacOS with labelme input to open its application

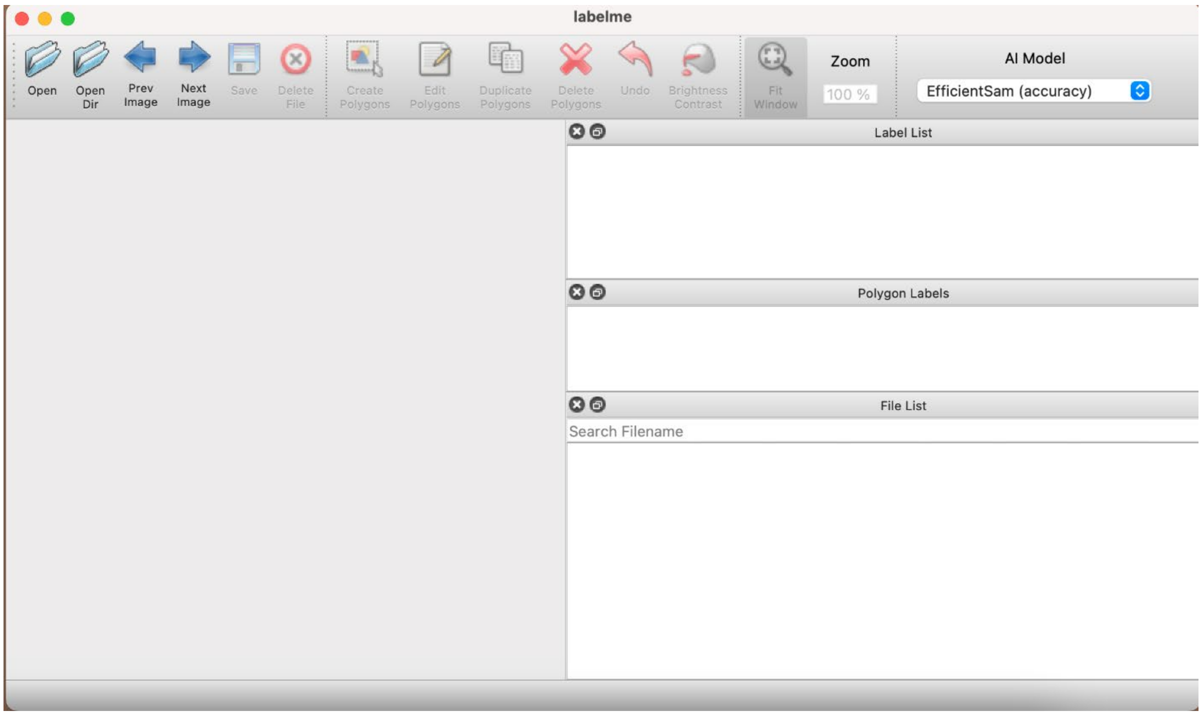


Figure 0.2: This is the label me interface in MacOS

Step 2: Each member loads directory with their name at root folder at Open Dir, then after that the list of images in that directory appears on the right bottom.

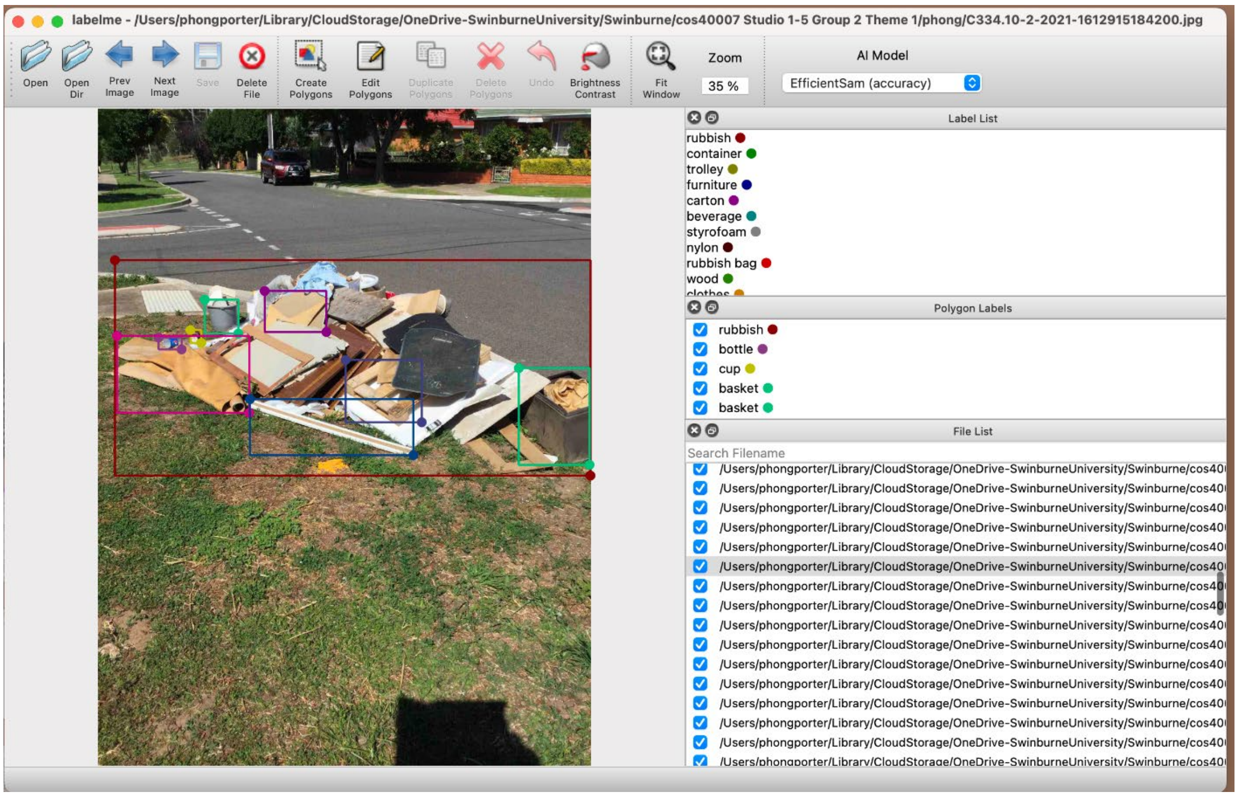
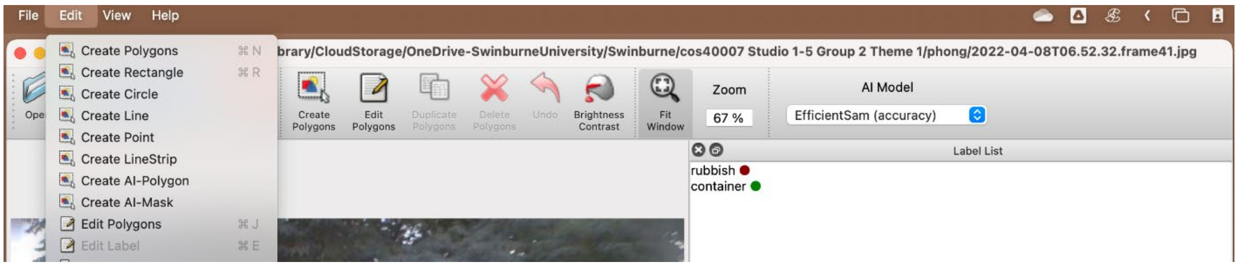


Figure 0.3: Open Dir is on the top left, adjacent to Open icon

Step 3: Choose Edit to choose the `Create Rectangle` for labeling or key combination in MacOS

Command + R or in Windows Ctrl + R



brary/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40007 Studio 1-5 Group 2 Theme 1/phong/2022-04-08T06.52.32.frame41.jpg

Zoom 67 %

AI Model EfficientSam (accuracy)

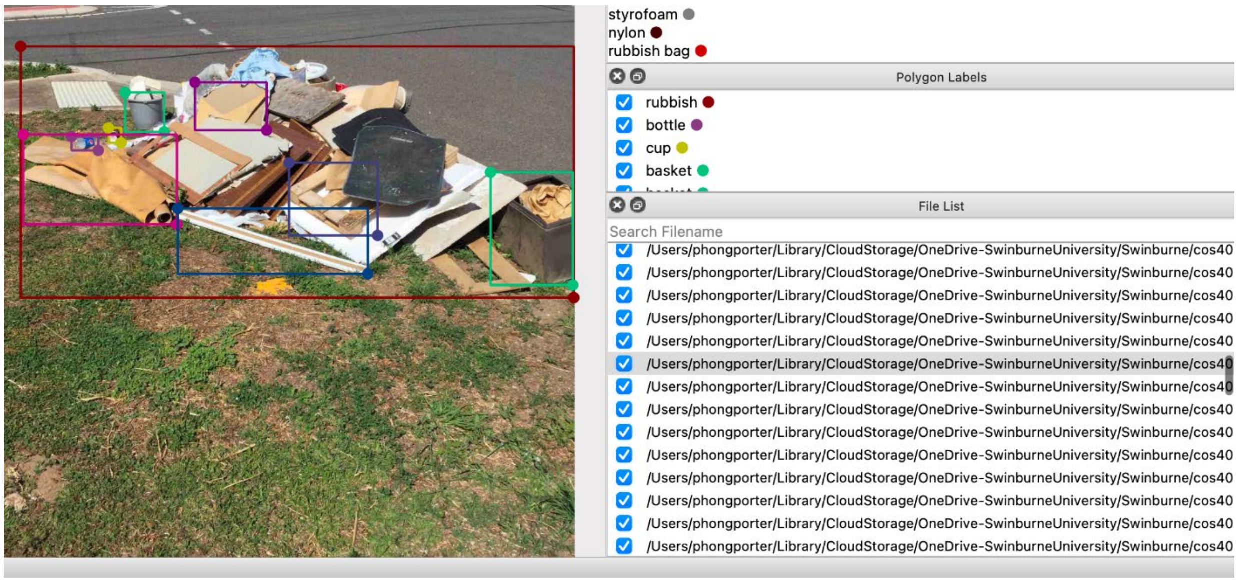
Label List

rubbish ●
container ●

Figure 0.4: The MacOS system shows the Edit at the top of the screen

Step 4:

A) Label the rubbish adjacent to each other, before label inside what it contains. For example, the red one surrounding all items called rubbish, and specific items inside that is two containers. After finishing, save the label JSON file at the team member's name folder same as the image that team member is working before continuing with other images.



styrofoam ●
nylon ●
rubbish bag ●

Polygon Labels

☑ rubbish ●
☑ bottle ●
☑ cup ●
☑ basket ●

File List

Search Filename

☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40
☑ /Users/phongporter/Library/CloudStorage/OneDrive-SwinburneUniversity/Swinburne/cos40

Figure 0.5: There is a tick at the image in the directory, shows that the image has a label JSON alongside that we have labelled before.

B) In the case that the image has many separate rubbish areas, the label will have different rubbish area with different items inside that.

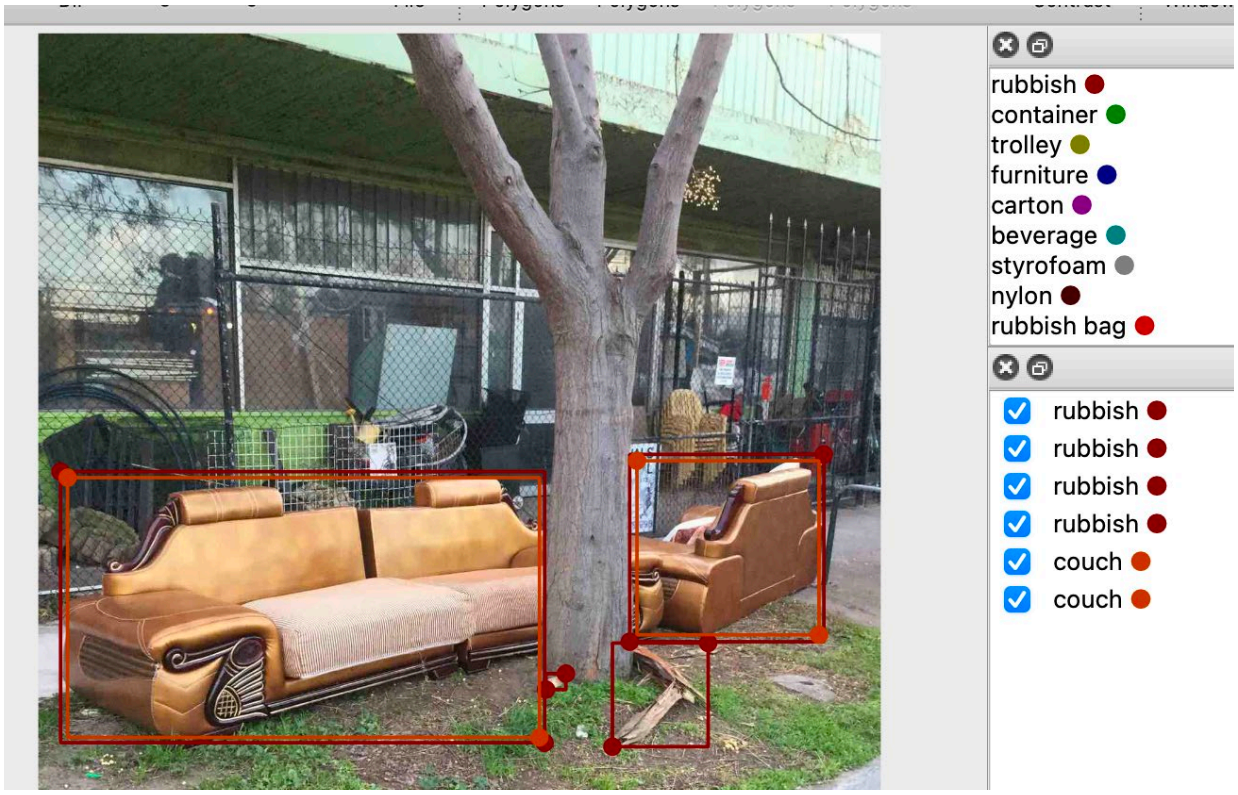


Figure 0.6: There are 3 rubbish areas with 3 different items inside that, including two sofas and wood

Step 5: Finishing image labelling, there are two ways to split the image into the `rubbish` and `non-rubbish` at `processed_data` at the root folder

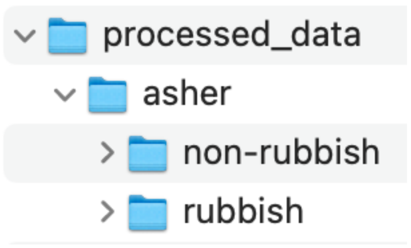
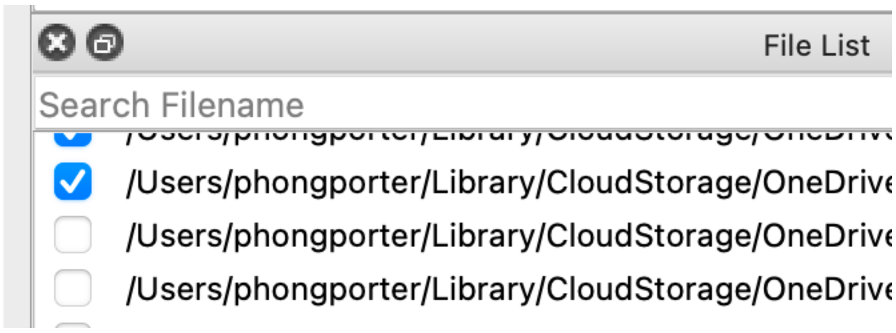


Figure 0.7: This is `asher` folder on `processed_data`, which is done after labelling images.

A) Organise manually: By looking at the labelme application on the directory images which one is not tick, means no rubbish in that image, we move them into the `non-rubbish` folder, and the rest of the rubbish images move into the `rubbish` folder. However, this old-fashioned way will be time-consuming and might mistakenly put the rubbish into the non-rubbish folder.


File List

Search Filename

/Users/phongporter/Library/CloudStorage/OneDrive

/Users/phongporter/Library/CloudStorage/OneDrive

/Users/phongporter/Library/CloudStorage/OneDrive

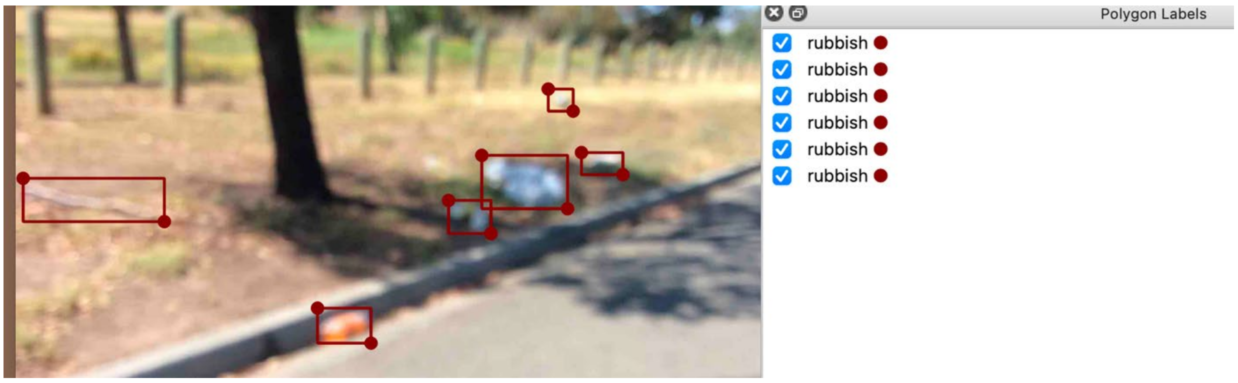
Figure 0.8: The File List section at the bottom right, which shows that tick means rubbish, and no-tick means non-rubbish

B) Using code provided by team leader: This is the most useful way for us to organise the data into `rubbish` and `non-rubbish` as first variable is the original directory, and second variable is the `non-rubbish` target directory. After moving all `non-rubbish` targets into the new directory, we will move the `rubbish` images into the new directory in the same level with `non-rubbish` folder. The filtering code script is in the Appendix section.

That is the step-by-step on how we did on labelling image.

2.2.4 Limitation:

Several rubbish positions with items inside are blurred due to the time of day or due to weather conditions, so this might hard to label accurately with our judgment. The provided image has several items inside the rubbish are said to be very difficult for us to fully recognize.


Polygon Labels

rubbish ●

rubbish ●

rubbish ●

rubbish ●

rubbish ●

rubbish ●

Figure 0.9: An example of the difficulty in finding out what item is inside the rubbish

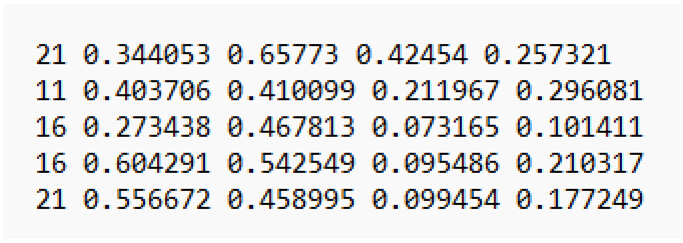
When labelling the image, there are some mistakes in the team that one or more members tried to label the item without having to label the "rubbish" for any image with rubbish or typo input, like `wood\` or `tire` (tyre is correct)

3. AI model development

3.1 Feature engineering / Feature extraction / Image processing

3.1.1 YOLO format

YOLO requires labels to be in ".txt" files. Since we labelled data using *labelme*, which format label in JSON, we have to convert that JSON into text files. For this task, we use labelme2yolo (https://pypi.org/project/labelme2yolo/). For each image with rubbish, there will be one corresponding text file, in which every line has the label index and the bounding box coordinates. The dataset.yaml file is also generated, indicating the folder structure (train/val folder directories) and the list of labels presented.



```
21 0.344053 0.65773 0.42454 0.257321
11 0.403706 0.410099 0.211967 0.296081
16 0.273438 0.467813 0.073165 0.101411
16 0.604291 0.542549 0.095486 0.210317
21 0.556672 0.458995 0.099454 0.177249
```

Figure 1.1. YOLO label file

As you can see from the sample, the first integer indicate the label. The following two float numbers represent the coordinate x and y of the bounding box, and the last two float represent the width and height of the bounding box.

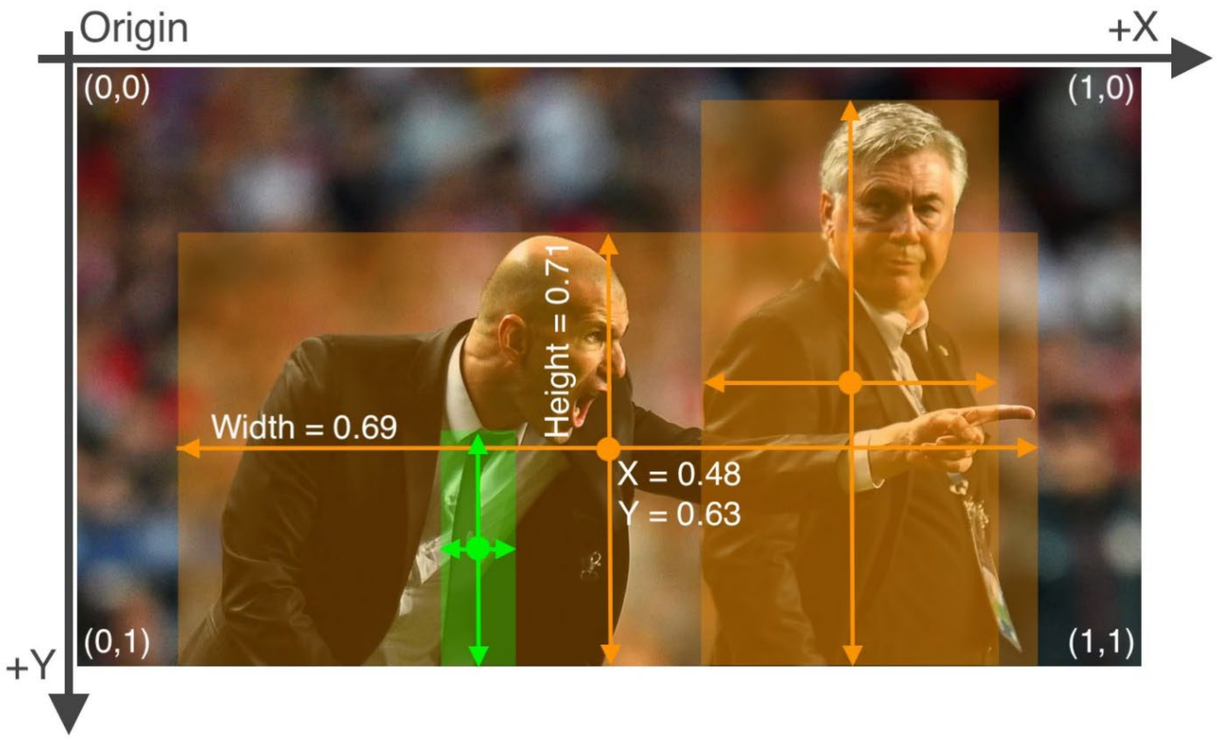
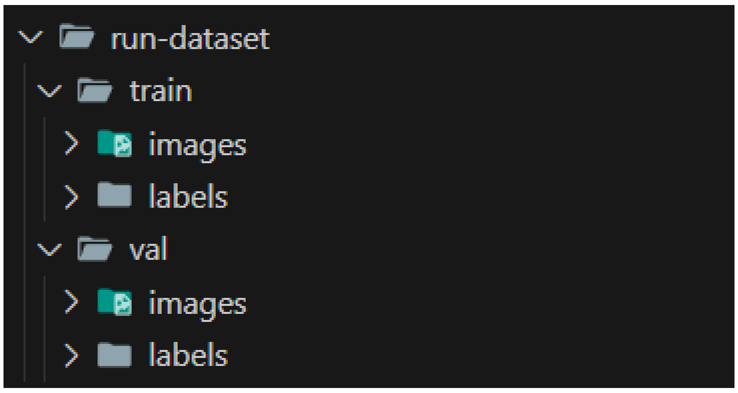


Figure 1.2. YOLO label format (image from https://docs.ultralytics.com/yolov5)


```
run-dataset
  train
    images
    labels
  val
    images
    labels
```
Figure 1.3. YOLO folder structure

Images and Labels folders must be organized like this to adapt to the train model script.

3.1.2 Segmentation mask

To train Unet, the dataset must include segmentation masks. Specifically, each image is complemented with a segmentation mask. In computer vision, a segmentation mask is the result of image segmentation algorithms that classify and "masked" each pixel in an image based on its category or item label. Usually, a colored or grayscale overlay on the underlying image, the segmentation mask distinguishes and divides various areas of the image, such as objects or regions, according to their borders. There are two types of segmentation mask:

- Semantic Segmentation: Assigns a class label to each pixel without distinguishing individual object instances. For example, all pixels in an image belonging to "cars" are labeled as "car."

- Instance Segmentation: Distinguishes between instances of objects, so each car in a scene would be labeled separately.

The dataset utlised for training Unet is converted from *labelme* to Pascal VOC (Visual Object Classes) format, which uses semantic segmentation. As the problem that the team solving is a multi-class exercise, the masks are coloured instead of being in grayscale. Each colour represents a unique class. There are a total of 31 object classes, so the dataset has 31 different colors. The code script for converting *labelme* to VOC is in Appendix section. This script is provided by the *labelme* repository.

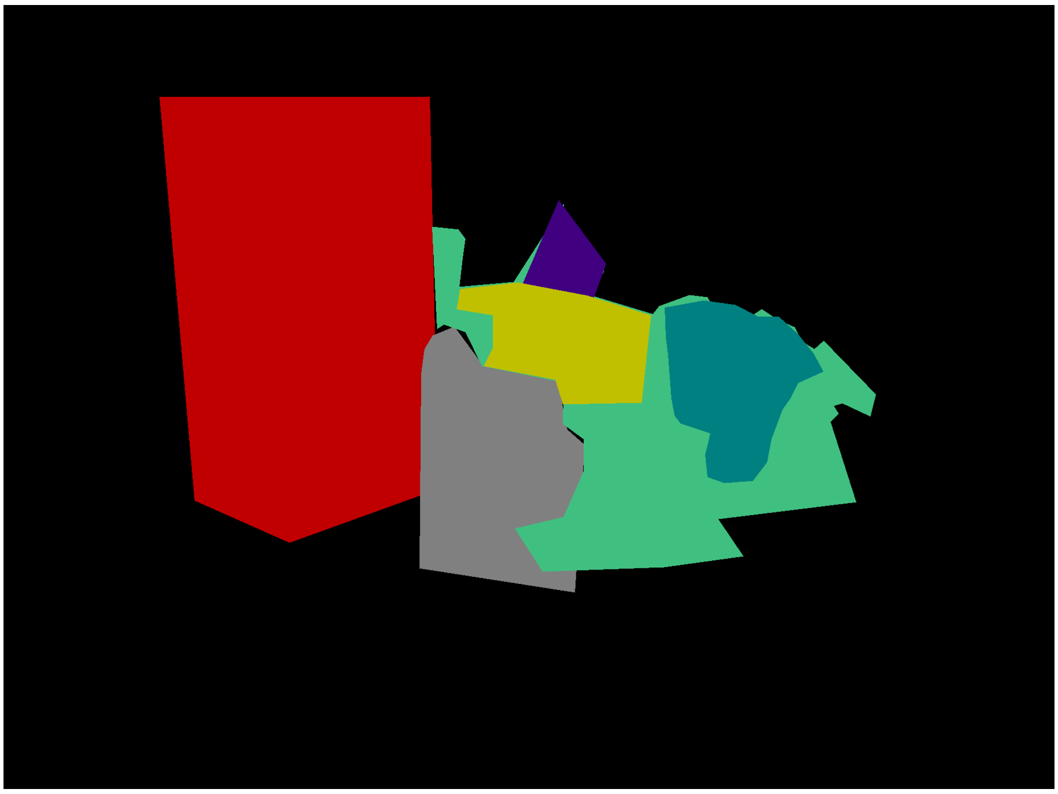


Figure 1.4. Example of a segmentation mask

3.2 Train/Test split

3.2.1 YOLO split

To train YOLO model, out of ~800 rubbish images, we split them with ratio 80-10-10 for training-validation-testing. The training-validation ratio is 90-10, which is a popular ratio, especially when we have limited data. We also add non-rubbish images into the dataset with the same ratio for training, validation and testing. Due to limited data, we only take ~80 images for testing.

3.2.2 Unet split

Similar to YOLO, the default training-validation split for Unet is also 90-10 (726 training images, 80 validation images). It is necessary to train Unet on a large training set to get the highest result, and 80 images are adequately diverse for validation. However, unlike YOLO, Unet requires segmentation mask for every training image, which non-rubbish images do not have. Thus, non-rubbish ones were not considered for training.

3.3 Training model

In this project, the team had two training iterations, both of which use around 800 images. In the first iteration, the dataset has too many classes (nealy 150 classes), so the performance of models after training is relatively low. For that reason, in the second iteration, the team decided to narrow the classes down to 31. The second iteration has shown significantly better outputs from models. Both model use PyTorch for training and data loading.

3.3.1 Training YOLO

YOLO model is taken from the Github repository of Ultralytics (https://github.com/ultralytics/yolov5). The settings applied are:

- Model size: yolov5l
- Image size: 640x640 (YOLO augments the images and labels automatically)
- GPU: GTX 1060
- Freeze 10 backbone layers.
- 50 epochs.
- Batches of eight.

The Ultralytics team provided Python script to train the model. Below is the terminal command to start training:

```
python train.py --img 640 --epochs 50 --data dataset.yaml --weights yolov5l.pt --freeze 10 --batch 8
```

3.3.2 Training Unet

The Unet model is taken from another Github repository (https://github.com/milesial/Pytorch-UNet.git). The below is the settings of model:

- Image scaling ratio: 1.0 (use original size)
- GPU: A100
- Learning rate: 1e-5 (0.00001)
- Use mixed precision (Mixed precision is a technique in deep learning where both 16-bit and 32-bit floating-point formats are used for model parameters to increase computational efficiency and reduce memory usage without significantly sacrificing model accuracy)
- 5 epochs
- Batch size: 1

The Github repository provides a script for training. This is the command to initialise the training:

```
!python train.py --amp
```

Here, the flag “--amp” is to enable mixed precision training. All the other settings have been set in the training script.

3.4 Evaluation of AI model

3.4.1 <u>IoU and confidence score</u>

IoU (Intersection over Union) is an efficient performance metric to assess how well the object detection model do its task. The metric represents how much overlapping area there is between prediction and ground truth bounding boxes. IoU of two images is calculated as the area of the intersection between 2 images, divided by the combined area of both images. On the other hand, confidence score of a prediction shows how certain the model is. With our model, most predictions have confidence score over 0.35, some reach 0.9. Below is the distribution of IoU values of the test data:

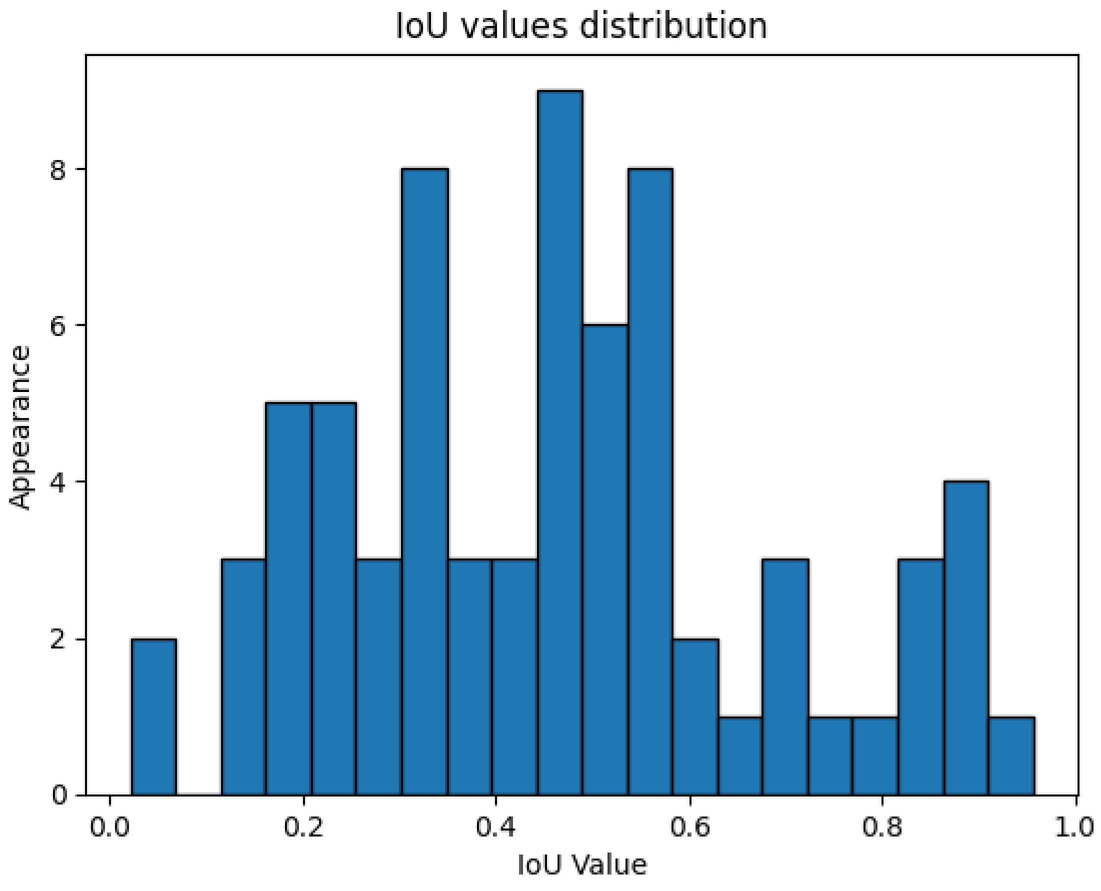


Fig1.5 IoU values distribution

We can see that due to the nature of rubbish, different types of rubbish are normally stacked on top of each other. Sometimes there are 2 labels in the ground truth label, but the model might predict them as 1 common label "rubbish". Average IoU score shows poor accuracy in those scenarios. However, if we look at the predicted images, the bounding boxes are relatively correct, even for the low IoU images.

We also calculate IoU for each type of rubbish separately (considering only true predictions).

Label	Average IoU (%)	Max IoU (%)	Object Type
0	84.34	88.85	waste-paper
1	82.07	88.85	carpet
2	74.80	95.23	rubbish bag
3	74.80	95.23	toy
4	77.04	95.23	container
5	77.04	95.23	nylon
6	77.04	95.23	accessory
7	67.34	95.57	carton
8	67.34	95.57	tyre
9	66.85	97.23	furniture
10	69.10	97.23	mattress
11	69.10	97.23	electronics good
12	69.10	97.23	trolley
13	69.10	97.23	transportation
14	69.10	97.23	rubble
15	71.44	97.23	chair
16	70.21	97.23	wood
17	62.28	97.23	rubbish
18	63.12	97.23	beverage
19	62.79	97.23	clothes
20	62.79	97.23	scrap metal
21	63.75	97.23	board
22	63.75	97.23	kitchenware
23	63.75	97.23	bag
24	63.75	97.23	rack
25	63.75	97.23	tarpaulin
26	63.75	97.23	table
27	63.75	97.23	plastic cup
28	64.35	97.23	sofa
29	64.18	97.23	plant
30	64.18	97.23	styrofoam

Fig1.6 IoU of different labels (true prediction only)

Below is an example image showing rubbish and carton label is predicted correctly, but one wrong prediction makes the IoU drop to 0.67.

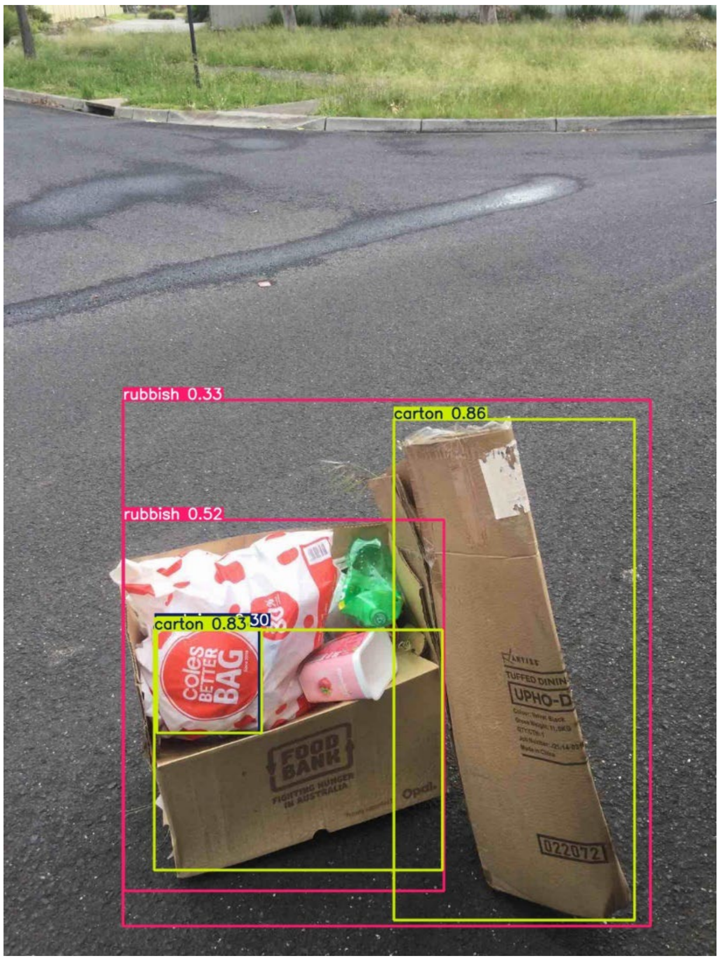


Fig1.7 Example

3.4.2 Dice score

The Dice score is the evaluation metric of Unet. It is recorded on *wandb* – a library to visualise deep learning models performance during training. Dice score is calculated as follows:

$$\text{Score} = \frac{2 * |A \cap B|}{|A| + |B|}$$

While A is the number of pixels in the predicted segmentation mask, and B is number of pixels in the ground truth mask. "A $\cap B$" represents the number of common pixels between A and B. The images below show the Dice score evaluated for each training steps.

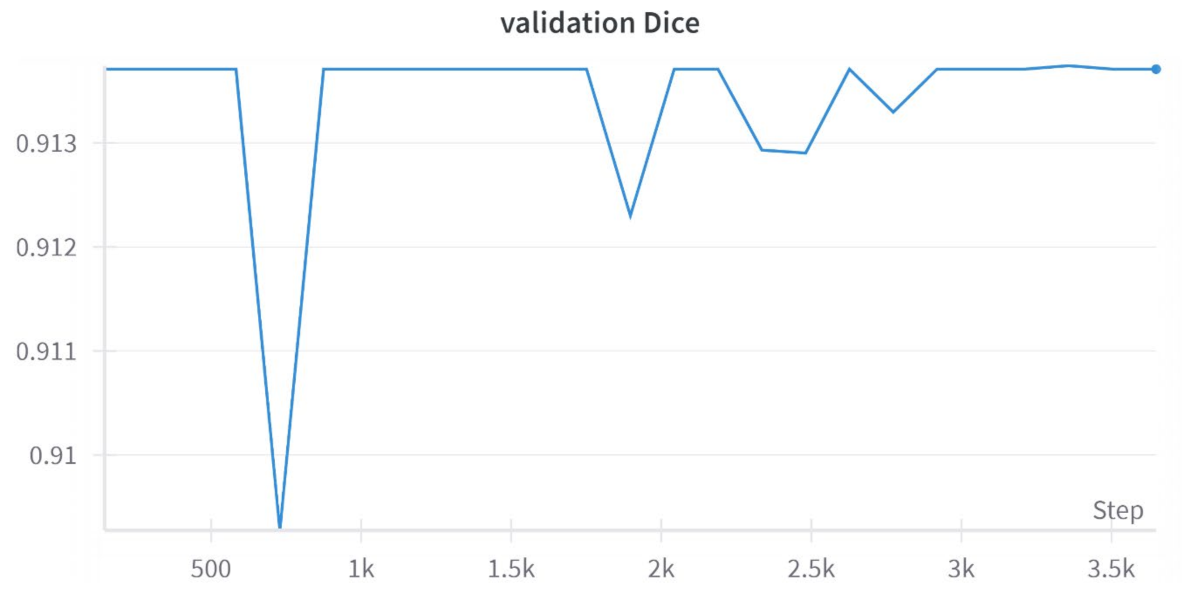


Fig1.8 Dice score

Although the averaged Dice score is relatively high throughout training (around 0.91), the predictions of Unet are flawed. Most of its predictions produce all-black mask, meaning that it can not detect the rubbish and specific items inside. Unet is primarily designed for image segmentation, which focus on producing pixel-level boundaries around objects and labelling each pixel in an image according to the object class it belongs to. Unet did poorly on identifying objects in the image where objects lie in a specific location rather than the whole image. This is why YOLO is preferrable for object detection, especially when it comes to the application of smart city.

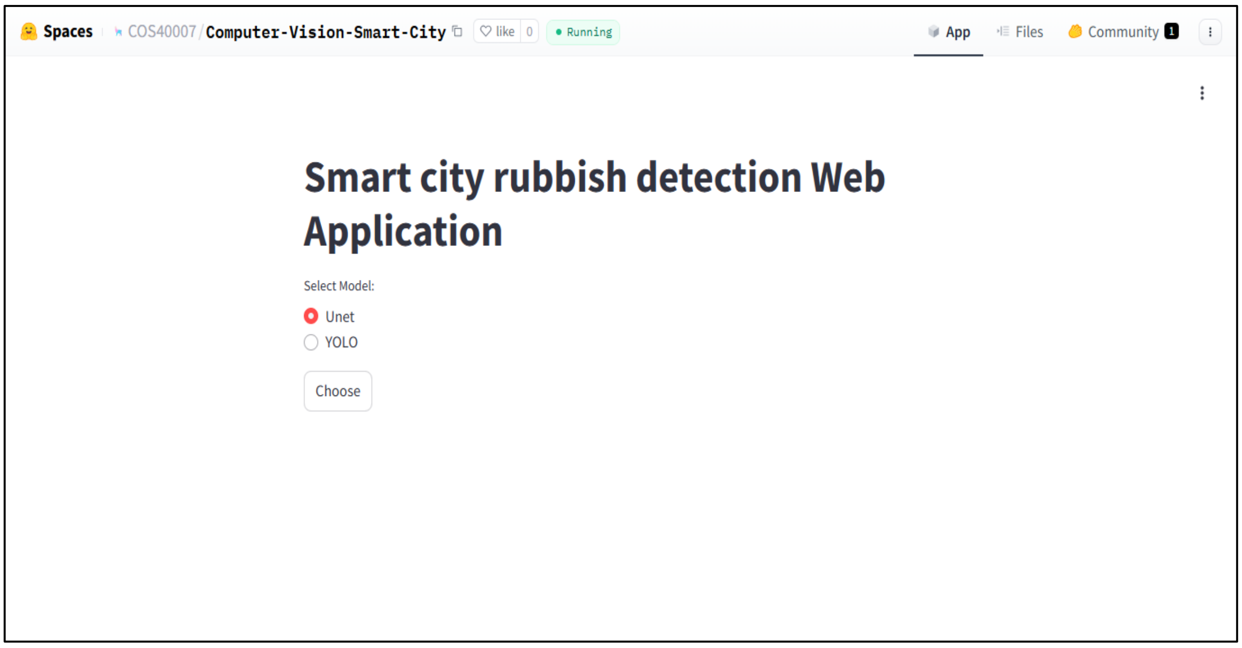
4. AI demonstrator

The AI demonstrator that our team is using for this project is a web-based application using Streamlit as the library for creating UI and Hugging Face for hosting UI. Users can use this interface to choose the type of labelling model (Unet and YOLOv5), enter objects to detect, and upload roadside images or videos with the purpose of detecting rubbish and identifying specific objects within those rubbish based on the objects that they listed.

For the development of the AI demonstrator, the webpage is built by constructing a Streamlit interface, whose purpose is to provide a user-friendly UI for choosing a labelling model and objects, uploading images or videos, and presenting outputs. Moreover, the backend of this interface was implemented with trained deep learning models (Unet and YOLOv5) to recognise rubbish and labelling objects within rubbish, such as mattresses, tables, and chairs. These models were trained on the images that our team took from the "rubbish" folder.

For the input and outcome to detect rubbish and specific items, users can choose either image files (JPG, JPEG, and PNG files) or video files (MP4, MOV, AVI, and MPEG4 files) to upload. Once uploaded, the file will be processed by the chosen model (Unet or YOLOv5) to detect rubbish and the items defined in the Classes field. After the analysis, the webpage will show the detection result with the list of the rubbish items and the image output that has the items labelled with bounding boxes around each item and a confidence score for each item.

Below are screenshots of the webpage:

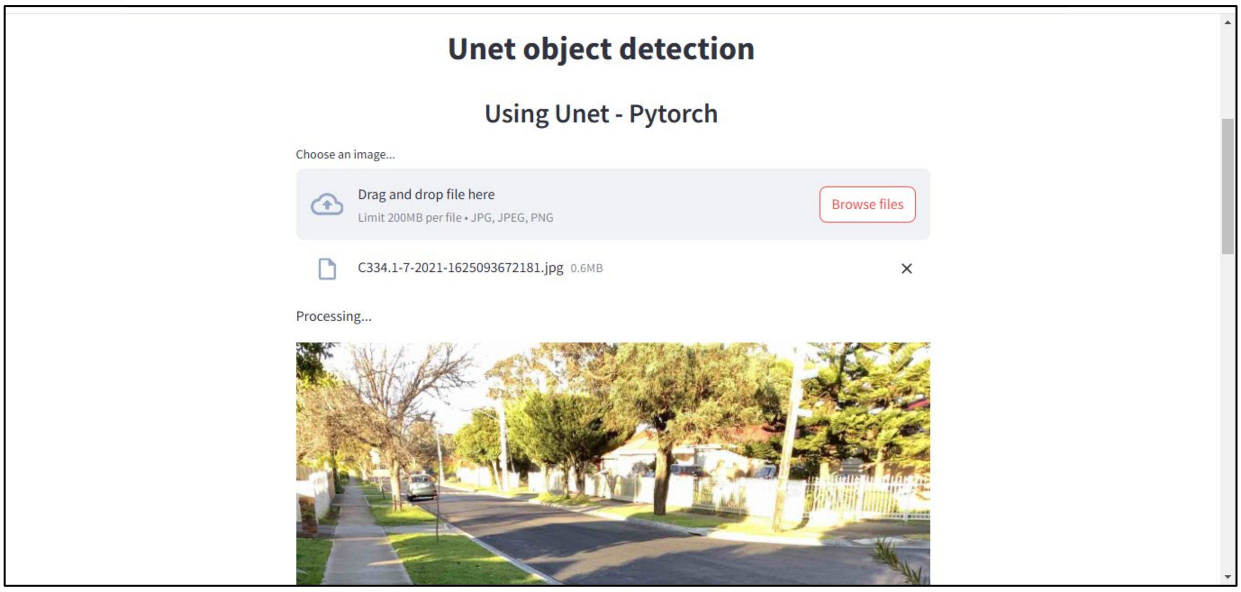


Smart city rubbish detection Web Application

Select Model:

[✓ Unet] YOLO

Figure 2.1: Home page to choose the model



Unet object detection

Using Unet - Pytorch

Choose an image...

C334.1-7-2021-1625093672181.jpg 0.6MB

Processing...

Figure 2.2: Unet model page

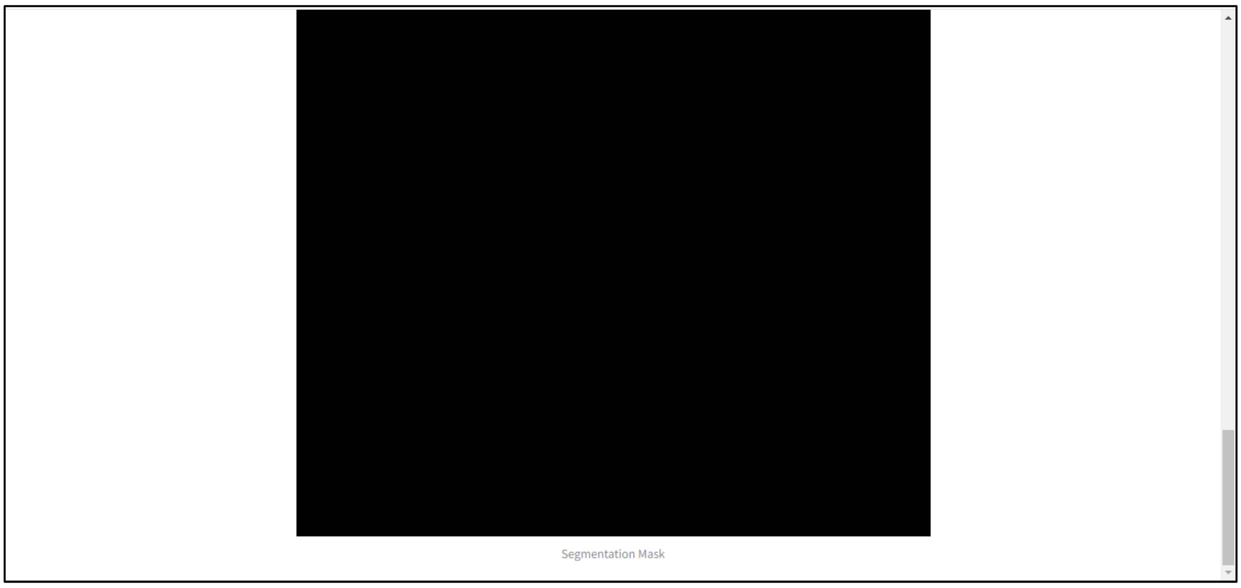
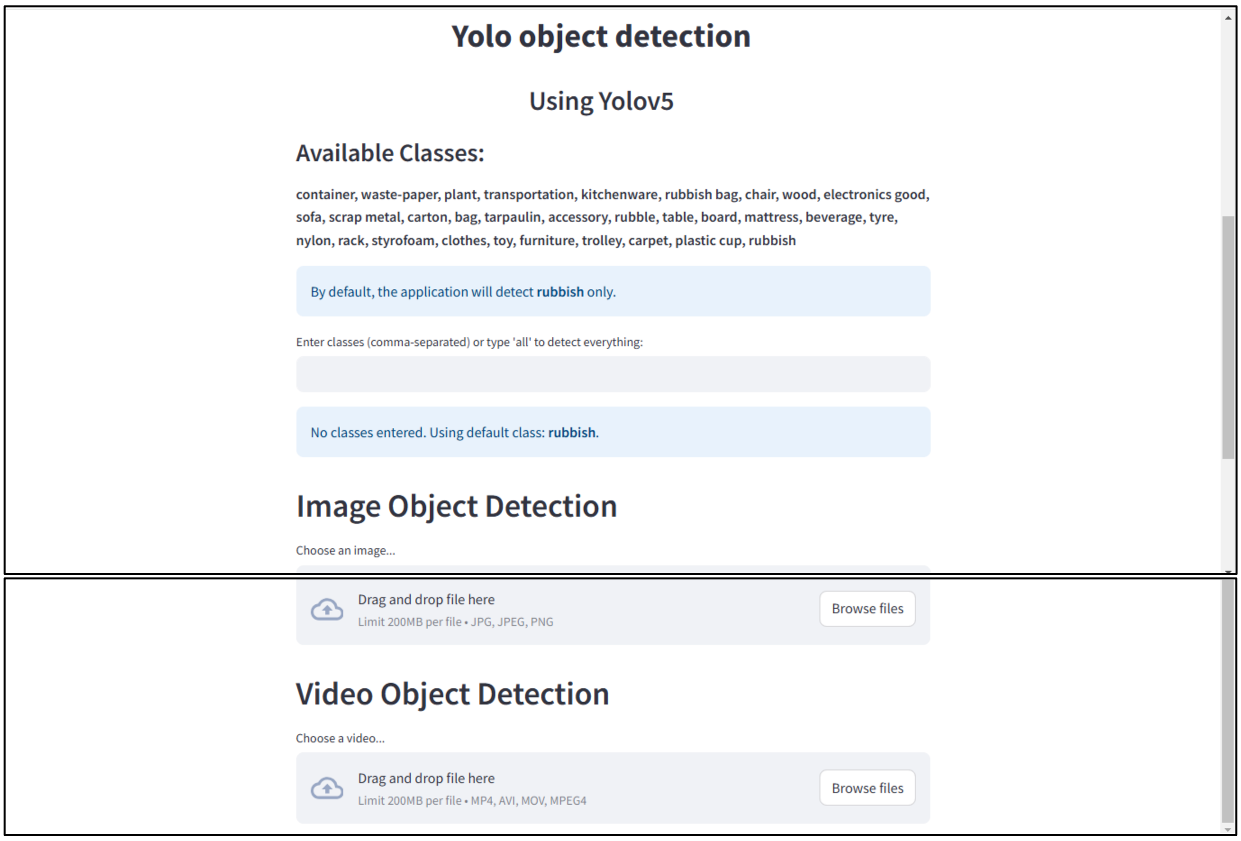


Figure 2.3: Result from the uploaded image in the Unet page



Yolo object detection

Using Yolov5

Available Classes:

container, waste-paper, plant, transportation, kitchenware, rubbish bag, chair, wood, electronics good, sofa, scrap metal, carton, bag, tarpaulin, accessory, rubble, table, board, mattress, beverage, tyre, nylon, rack, styrofoam, clothes, toy, furniture, trolley, carpet, plastic cup, rubbish

By default, the application will detect **rubbish** only.

Enter classes (comma-separated) or type 'all' to detect everything:

No classes entered. Using default class: **rubbish**.

Image Object Detection

Choose an image...

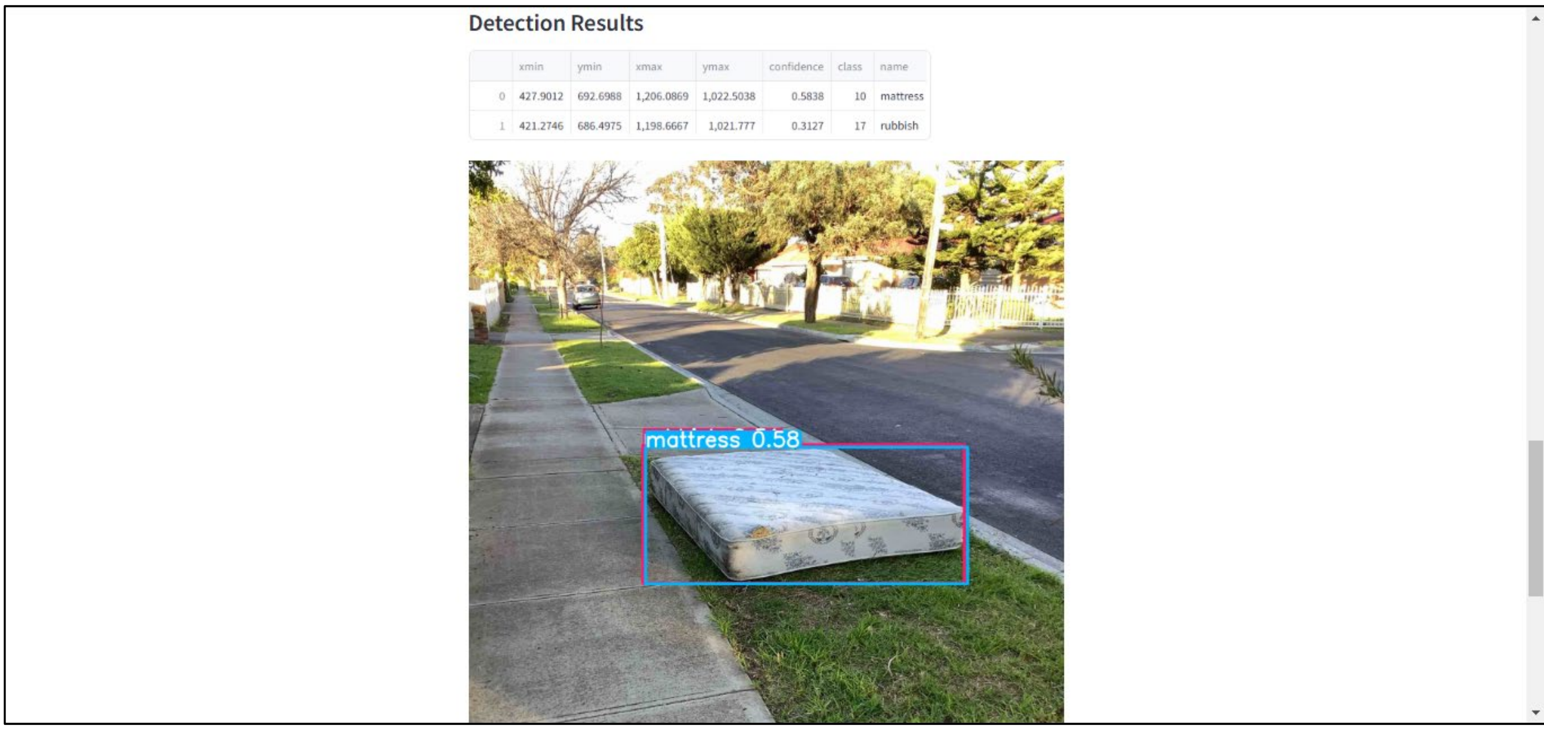
Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Video Object Detection

Choose a video...

Drag and drop file here
Limit 200MB per file • MP4, AVI, MOV, MPEG4

Figure 2.4: YOLO webpage with the classes input field and file uploading spaces



Detection Results

	xmin	ymin	xmax	ymax	confidence	class	name
0	427.9012	692.6988	1,206.0869	1,022.5038	0.5838	10	mattress
1	421.2746	686.4975	1,198.6667	1,021.777	0.3127	17	rubbish

Figure 2.5: Output of the uploaded file in the YOLO page

5. Conclusions

Overall, this project utilises Unet and YOLOv5 models to detect rubbish and specific items within rubbish item in roadside image data that are labelled using LabelMe's bounding box and polygon techniques. The outcome of this project includes identifying the specific type of detected rubbish (mattress, furniture, chair, etc) which can be useful for city council to plan on dealing with these types of rubbish. However, our team has faced some technical challenges in the process of developing this project, such as consuming a lot of time on labelling data, which can be managed by using annotation tools, or ensuring consistency when labelling data, which can be solved by setting up some requirements when labelling. Finally, throughout this project, our team has learned about the importance of data labelling in achieving high precision in object detection tasks. We believe that this project can help our team broaden our knowledge by applying what we have learned in this semester - from processing data to training models – in a real-world setting.

References

1) OpenAI. (2024, Oct 8th). "How to build an user interface for YOLO using Streamlit" [ChatGPT response]. Retrieved from https://chat.openai.com/
2) Redmon, J, Divvala, S, Girshick, R & Farhadi, A 2016, *You Only Look Once: Unified, Real-Time Object Detection*, 9 May.
3) Ronneberger, O, Fischer, P & Brox, T 2015, *U-Net: Convolutional Networks for Biomedical*

Image Segmentation, 18 May.

4) Samar K 2024, *Building an Object Detection Web App with YOLO-World and Streamlit*, Medium, viewed 31 October 2024, <https://medium.com/@samarkaland/building-an-object-detection-web-app-with-yolo-world-and-streamlit-911be7571185>.
5) Streamlit 2024, *30 Days of Streamlit*, Streamlit.app, viewed 31 October 2024, <https://30days.streamlit.app/>.

Appendix

- Notebook to train Unet: https://colab.research.google.com/drive/1xtm2gXHmkb1TNU8ItqFu5sjSZJEXuAiO?usp=sharing
- Notebook to train YOLO: https://drive.google.com/file/d/1eWKM5xWL7_vJ4tmUPH-nBPke4s3Txv-c/view?usp=sharing
- Code for conversion from labelme to VOC: labelme2unetPNG.py
- Non-rubbish filtering code: filter.py
- Processed data with labelme annotations: processed_data. Each member is responsible to label a subset. Folders with "_p2" suffix are results of second iteration. "Final_data.zip" is the final synthesised dataset.
- Final models' weights: checkpoints
- AI demonstrator: https://huggingface.co/spaces/COS40007/Computer-Vision-Smart-City