SFU

SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

ENSC 482: Final Project
Recyclable Waste Detector and Classifier

August 15th, 2023

Name	**Student #**	**Contribution(%)**
Tao Li	301342974	30
Flynn Dowey	301422413	40
Steven Borkowski	301347154	30

Abstract

This report describes the development and implementation of a recyclable waste detection and sorting system. Using the YOLO (You Only Look Once) object detection model, the system can accurately identify between two different waste categories: organics and . By integrating the advanced features of Tensorflow, the subsequent object classification process can be facilitated. The entire system is seamlessly integrated into a Graphical User Interface (GUI) using Python, OpenCV, and Unity, enhancing user accessibility and interactivity. The successful fusion of cutting-edge technologies such as detection, sorting and user interface design highlights the potential of the system to make a significant contribution to streamlining the recycling process, thereby promoting sustainable waste management practice.

Table of Contents

1 Introduction

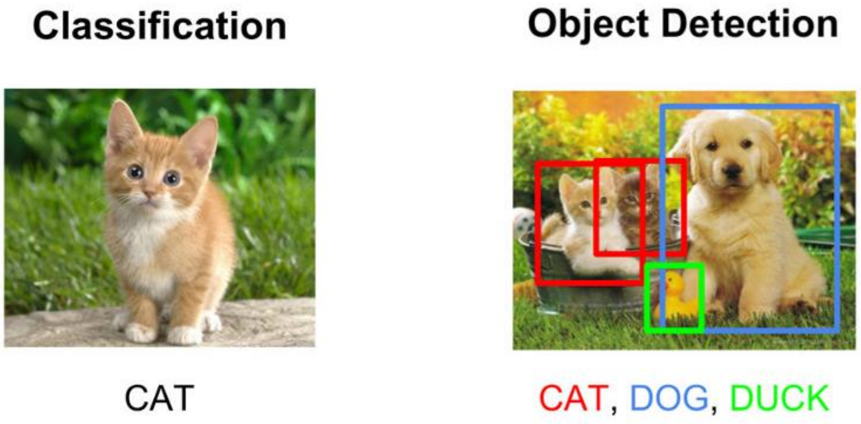
In an era of growing environmental concerns and increasing waste volumes, the need for efficient waste management solutions has never been more pressing. This project embarks on a journey to address this urgent need by introducing a waste detection and sorting system. By leveraging the capabilities of the YOLO (You Only Look Once) object detection, the system aims to sort waste accurately by identifying it as recycling or organic. The integration of Tensorflow further facilitates the subsequent object categorization process. Integrated into a user-friendly graphical user interface (GUI) powered by Python, OpenCV and Unity, the system not only improves waste management efficiency, but also promotes sustainable practices by encouraging optimal recycling. The integration of cutting-edge technologies such as detection, sorting, and interactive user experience highlights the system's potential to revolutionize the recycling process, fostering the future of waste management.

2 Background Materials

The Difference Between Classification and Object Detection

Simply put, object classification is a machine learning task where a model must distinguish and image, data, or any similar phenomenon into k sperate classes. For example, consider the machine learning task of separating spam emails from non-spam emails. In this case, there are two classes: the email is spam, or the email is not spam. The model must decide out of n emails whether each is spam or not spam. For our project this classification problem is similar, is the object recyclable or compostable.

In object detection the task changes, now the model is given k different classes and its job is to locate the ith classes within a sample space of data and it produce a bounding region to sperate this identity from the rest of the data. The figure below summarizes the differences between classification and detection [1].



What is Transfer-Learning?

Transfer learning is a powerful tool in the machine learning field, it allows designers to use pretrained weights from large and accurate models to reduce the need to train the entire network over a new dataset. Typically, you “freeze” all layers of the models, except for the last layer where you can modify the number of classes or add an architecture of your own.

2.1 Python

Python is a widely used programming language that formed the cornerstone of the development of this project. Its ecosystem of libraries and frameworks allows it to be adapted to a wide range of applications. In this project, Python was used for model integration, algorithm implementation, and overall system orchestration.

2.2 YOLOv4

The YOLO object detection model is a powerful computer vision and machine learning tool. Known for its ability to process images with extreme accuracy in real time, YOLO plays a key role in the project's waste detection component. Its fast-processing capabilities, coupled with the ability to simultaneously predict object classes and bounding box coordinates, are critical for identifying and sorting recyclable waste such as cans, bottles, and glass.

The Yolo project consists of various machine learning models, ranging from yolov2, yolov3 and yolov4 also called darknet. These models are quite large and the average training time for the COCO dataset could vary between hours or days depending on your computers processing power. For this project we will be using a modified version of the Yolov4 model, called Yolov4 tiny, which has less models’ parameters which is better suited for training on the ordinary computer. A summary of Yolov4’s parameters are listed below [2].

	FPS	mAP	GPU (MB)	FLOPs
Yolov4 tiny	270	38.1	1055	6.8

2.3 YOLOv5

Yolov5 is a sperate project from Yolov4 or darknet. It offers various sizes of object detection, segmentation, and classification machine learning models. We only used the nano and small models, denoted as Yolov5n and Yolov5s respectively. Unlike Yolov4, the models in Yolov5 are quick to train and give suitable results. A summary of model parameters is listed below.

	Image size (pixels)	mAP	CPU speed (ms)	FLOPs
Yolov5n	640	48.7	0.6	4.5
Yolov5s	640	56.8	0.9	16.5

More information on the project can be found here [3].

2.4 TensorFlow & PyTorch

TensorFlow is a machine learning Python package. PyTorch, is another machine learning package available in python. TensorFlow will allow us to apply transfer learning to the classification model, MobileNetv2, and enable us to use it dynamically with the webcam.

PyTorch is responsible for loading the object detection models as they are only available on this platform.

2.5 OpenCV

OpenCV plays a pivotal role in this project. OpenCV has a library of functions and algorithms that enhance image and video analysis. Most notably, its live camera feed feature is integral for this project. By interfacing with cameras, OpenCV provides the real-time stream needed for waste detection.

2.6 Unity

Unity is a powerful game engine that extends its capabilities from game development to GUI creation. By integrating Unity, the project provides a visually appealing and user-friendly Graphical User Interface (GUI) that enhances user interaction with the system.

3 System Development

Object Detection

For the object detection portion of the pipeline, we will be using the TACO dataset [4]. The TACO dataset correlates well with our problem definition of classifying recyclables from organics. TACO is an open-source dataset consisting of 80 classes of 1500 annotated images of litter found all over the world. There will be no data augmentation used in our pipeline, although it can prove to be useful in training deep learning models. The data was split into and 80/20 train/test formation.

We summarized the parameters of the three models trained below.

Model	Batch #	Image size	epochs
Yolov5n	1	320x320	100
Yolov5s	1	320x320	100
Yolov4-tiny	32	640x640	6000

Classification

For the classification layer, we decided to use a pre-made model available from TensorFlow called MobileNetv2 [5]. We applied transfer learning to this as with the object detection models, using the ImageNet weights. At the output layer we added an average pooling layer which is then connected to two perceptrons: one for each class (organic or recycling). The perceptron at the output with the highest value will be activated. We present the training parameters below:

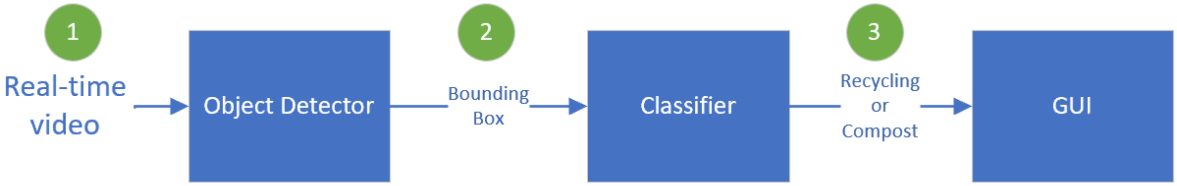
Image size	Batch number	Epochs
160x160	32	6

The dataset used for the classification layer is available on Kaggle [6]. As with the TACO dataset, we resort to the common 80/20 train/test split method.

The learning rate for each of the models was set to the default value for simplicity. Note that the Learning rate is an important parameter in machine learning as it can sway the convergence of a model's ability to predict reliably. We will exclude discussing the architecture of each model as this is beyond the scope of the paper. However, the principle of minimizing the cost function, $J(w, x, b)$, to achieve a local minimum still applies to all models.

The Pipeline

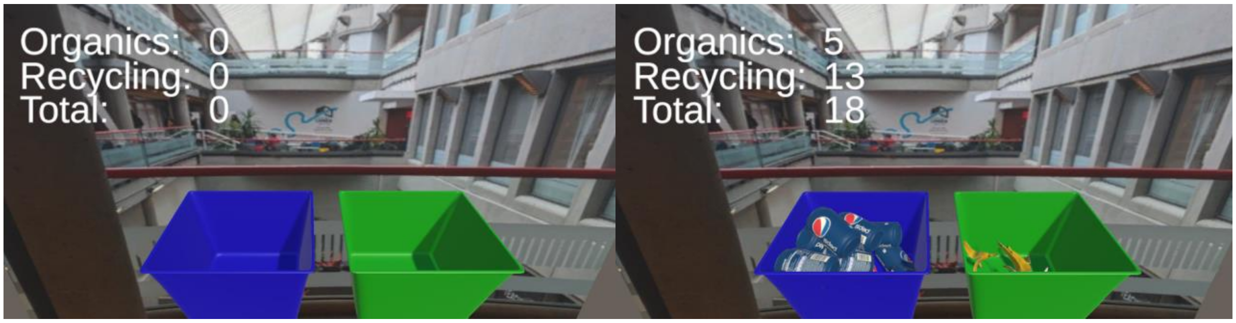
The system we designed has the following pipeline:



In step 1, the real-time video footage from the computer's web camera will be captured by OpenCV and subsequently sent to the object detector. In our case, the object detector is Yolov5s as this produced the best results. In step 2, the object detector will produce a set of coordinates which correspond to the bounding box surrounding the detected object. In step 3, the bounding box will be used to crop the image to exclude objected not considered of interest. The new cropped image is then fed to the classifier, which will produce a decision of either recycling or compost.

Unity

The graphical user interface was developed in Unity. A socket connection is established with Python over the device's IP address, through which data is sent upon detecting and classifying objects. The C# code listens on the port, and upon receiving the class, will instantiate a game object that falls in the corresponding bin. Additionally, the statistics of the current session are displayed on the left. The image below shows the game view:



4 Experimental Result and Analysis

As a preliminary, we need to define the following terms:

1. Precision: This is a performance metric; it defines the number of positive instances that were correct [7].

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

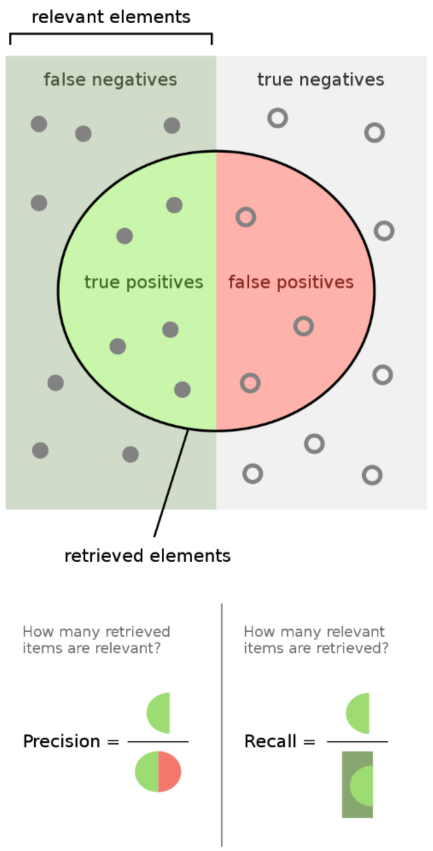
2. Recall: A performance metric; it defines the number of actual positives that were correctly identified [7].

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

3. F1 score: A performance metric; it measures the model's accuracy by combining precision and recall [7].

$$\text{F1 score} = \frac{2 \times \text{recall} \times \text{precision}}{\text{precision} + \text{recall}}$$

Graphically, we can summarize the above definitions [7].



Data and Number of Classes

As mentioned before, we are using the TACO dataset on the object detection models. Initially, we used the default number of classes, 80, and trained it on each of the object detection models. This method of including 80 classes proved to be costly, both in training time and performance. As a results, we decided to sub categorize the 80 classes into 4 manageable classes labeled as '0', '1', '2', '3'. The names of the classes were chosen arbitrarily but the point was to make it easier for the models to learn. Each of the classes held objects of similar traits, for example, class '0' would hold objects in the tuple {'plastic bottle', 'glass bottle', …} and class '1' would hold objects in the tuple {'plastic wrap', 'Styrofoam', …}. This method was applied to class '3' and '4'.

For the classification layer, we only have two classes which are ‘O’ – organic and ‘R’ – recycling. This dataset is simpler than TACO as the annotations are just the photos designated classes. You can find documentation relating to the dataset here [6].

Results for yolov4-tiny:

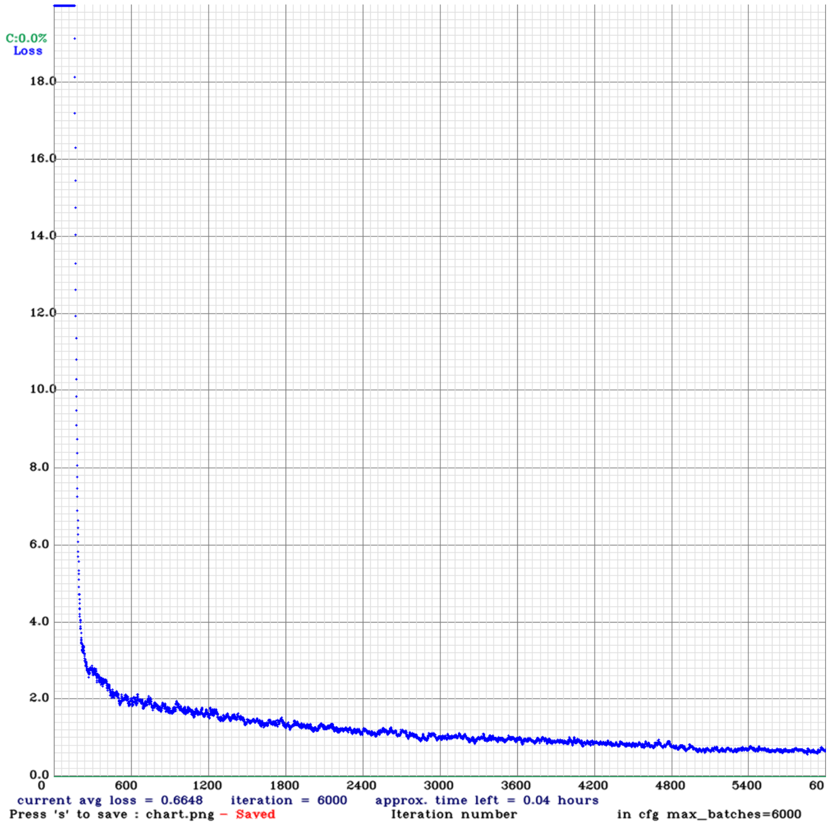
The table below summarizes the performance of the yolov4-tiny model. Refer to table 3 for the number of batches, epochs, and image size.

	Class ‘0’	Class ‘1’	Class ‘2’	Class ‘3’
Average precision	2.88%	16.06%	25.11%	1.23%

Aggregate performance:

Precision	23%
Recall	19%
F1 score	21%
IoU[1]	17.48%
Time to detection	9 seconds
Total training time	>24hrs

We present the loss vs epoch curve below.

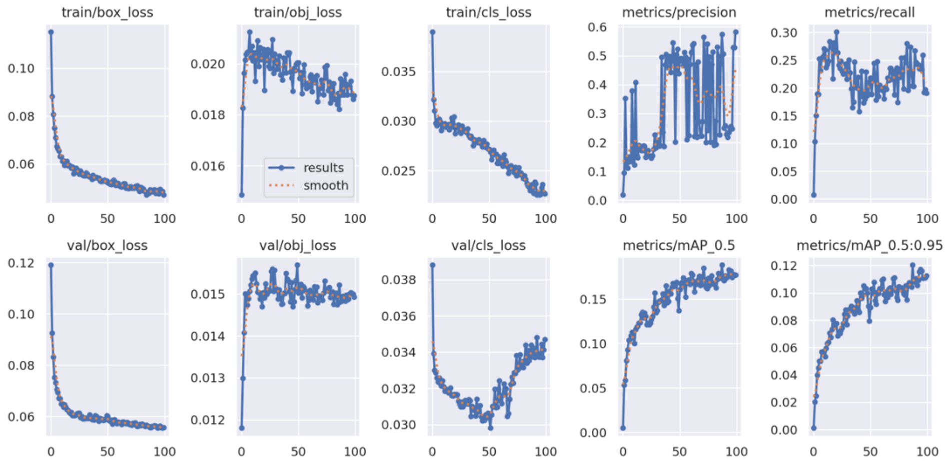


[1] IoU: intersection over union; how similar the predicted bounding box is with the ground truth bounding box.

Referring to the results above, we can clearly state that the model has room for improvement. The overall performance of the model was poor to say the least. Steps for improvement could include data augmentation, hyperparameter adjustments, or considering a larger model like yolov3 of yolov4. As a side note: training this dataset on yolov4 would take several days on the average GPU. Our machine did not have enough memory to support yolov4 even with low resolution and low number of batches. Considering Google Collab is an option, but the number of compute units is limited to 100 per month (100 compute units would be just enough to train yolov5n).

Yolov5n results

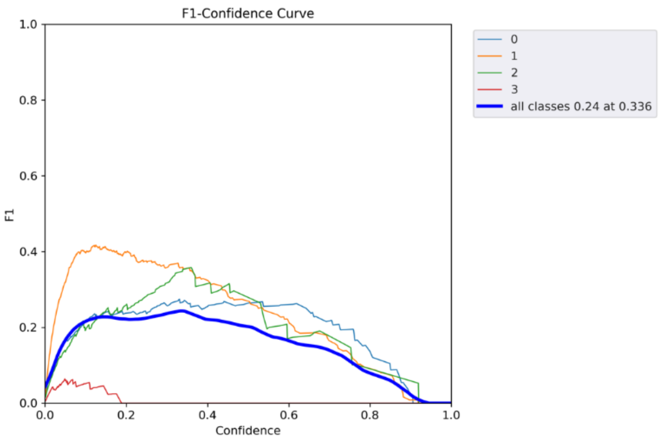
The following illustrate the performance of the smallest of the yolov5 family, yolov5n.



From the above figure we can deduce the following: training and validation for the bounding box loss both decrease exponentially indicating the model is learning without overfitting. The object loss is concerning as the validation loss remains stagnant at 0.015 after 100 epochs, while the training loss decreases. For the class loss we notice that the model begins to overfit after 50 epochs; this is evident from the increase in validation loss while the training loss continues to decrease. Fortunately, both the object and class loss are not relevant for us.

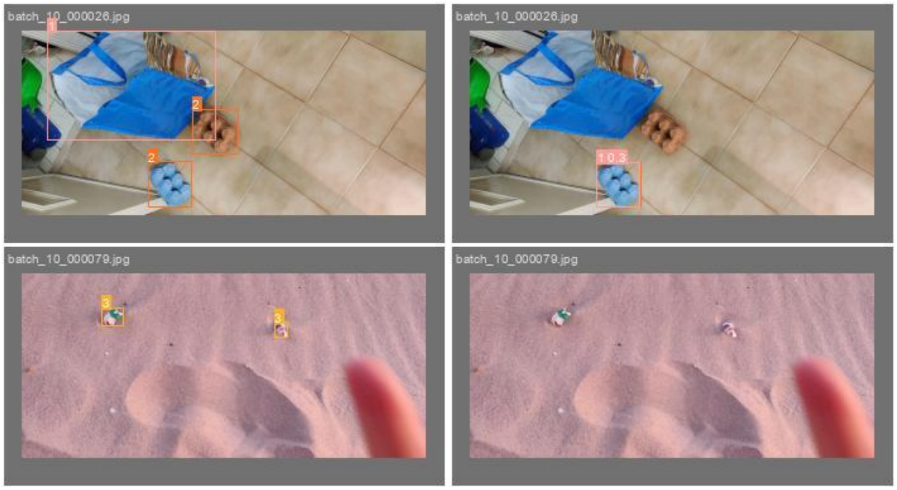
Continuing with the above figure, we notice that the precision tends to fluctuate between 60% and 20%, this could be from the size and position of the objects in the images; a very small object is hard to detect while a large one is easier. The maximum average precision recorded was around 17% to 18% which is worse than yolov4-tiny. Finally, the recall tends to remain around 30% to 15%. The total training time for yolov5n was around 10 hours.

In terms of F1 score we obtain the following:



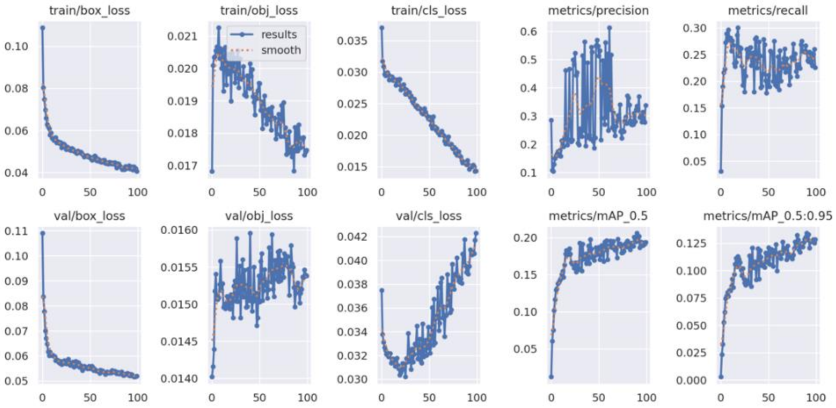
Notice that class ‘1’ performs the best at a confidence level of 0.1 while the aggregate is best at 0.336. These values are still poor, much like yolov4-tiny indicating further steps are needed to increase performance.

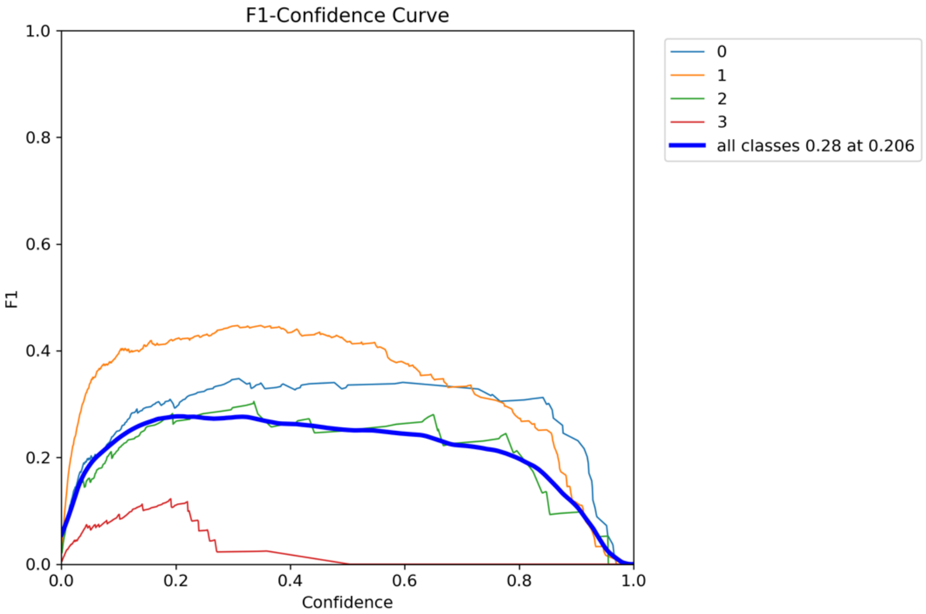
Below we present a sample of the prediction’s vs ground truth labels.



Yolov5s results:

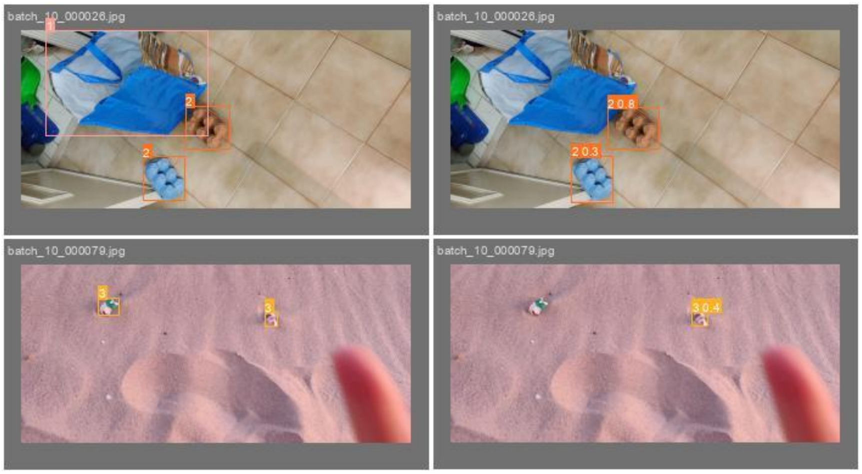
Like above, we will present the performance metrics of the second smallest model in the yolov5 family, yolov5s.





Much like yolov5n, we see that the model tends to overfit on the object and classes; this is evident from the rise in validation loss when training loss remains low. However, the box loss is showing the desired result of generalizing the data. In terms of precision and recall, we get the most optimal results with yolov5s with a precision of 32% and recall of about 25%. By no means are these metrics acceptable, they are rather very poor. Referring to the F1 curve, we see that yolov5s performs better than yolov5n, although this performance increase is marginal with the aggregated F1 value of 0.28 at 0.206 confidence. The training time for yolov5s was approximately 15 to 16 hours.

Below we present a sample of the prediction's vs ground truth labels.



MobileNetv2 results:

The results for MobileNetv2 are shown below.

Precision	Recall	F1 score
0.4512	0.4847	0.4673

These metrics are better than the object detection's results; however, these metrics indicate that the model has more room for improvement. Having a recall and precision value just under 50% indicated that we can correctly predict and classify about half of the data. Note that these values were produced on a separate subset of the training data that was not previously shown to the model. During training, the accuracy of the test and training data hovered around 80% which indicates that the model is unable to generalize.

To increase the performance, we could 'un-freeze' more layers of MobileNetv2, increase the number of images in the dataset or resort to a larger model.

5 Discussion

In this project, we set out to lay a groundwork for a waste sorter system with an intuitive user interface. The preliminary results shown here illustrate the methods used to develop the app and its robustness. Firstly, the variety of tools used were discussed and how they were integrated together. A YOLOv5s detector is cascaded with a classifier to combine the strengths of both. These machine learning models are integrated into Python using PyTorch and TensorFlow. By acquiring a live feed with OpenCV, the image is processed, and the results are displayed in Unity. Secondly, the app itself was introduced, including its GUI and the accuracy of the object detector. By improving its reliability, such an app could be used commercially or in schools to facilitate waste disposal.

From the results in section 4, more time needs to be dedicated to training an effective machine learning classifier and object detector. However, a large amount of computing power is required to achieve these results. The model that was developed in this paper has its limitations when not used in a plain background, as a result extraneous bounding boxes are created, and false classifications are made. This is caused by the ill-trained object detection model and use of only two classes at the output.

Darknet has great potential to be used effectively for this project, however, we return to the argument of GPU memory. If one had a limitless amount of computing power, this could be easily achieved. Yolov5 is a fast and efficient model, however, our computers could only handle up to the small version of yolov5.

The choice of not augmenting the data also played a part in the poor results of the model. It is a common practice to augment your data by rotating the images, flipping them, or adding filters. This artificially increases the size of your dataset and in turn increases your model's performance.

6 Future Improvement

Further development can make this project more robust and able to see wider use. Firstly, the object classifier can be trained on more classes, making it more robust. For training the detector, the TACO dataset was used. This is an open-source resource that has annotated images of waste in the wild [4]. Unfortunately, there are less than 10,000 annotated images in this dataset, so in future revisions of the project, a more extensive source can be used. Additionally, when training the model, a more top-down approach can be taken that involves guiding the model through iterations of the training and evaluation. Finally, in implementing the user interface, more objects can be used to represent the waste being sorted when more classes are added to the model.

When implementing this system in the future, the designer must consider their computing resources to implement a reliable model. The intent of the project was to classify recyclables vs organics in a public environment; thus, the use of a microprocessor is suggested. There are various processors on the market suited for machine learning such as the Nvidia Jetson which incorporates a small GPU on a microprocessor.

The designer's choice of data collection is also of importance for future work. Creating your own dataset in conjunction with TACO could improve the model's performance. Also adding a portion of the model to perform object segmentation could decrease the false-positive rate of the bounding boxes.

In conclusion, we can make the following statement: for accurate and efficient machine learning models, one will need great computing power and enough data. The field of machine and deep learning is growing at a great pace; thus, it is the designer's responsibility to be aware of the new technologies and methods presented.

References

[1] Datacamp, "A Beginner's Guide to Object Detection," [Online]. Available: https://www.datacamp.com/tutorial/object-detection-guide.

[2] L. Z. S. L. a. Y. J. ZICONG JIANG, "Real-time object detection method for embedded devices," 2020.

[3] Ultralytics, "Github," [Online]. Available: https://github.com/ultralytics/yolov5.

[4] P. Proenca, "Trash Annotations in Context," TACO, 2023. [Online]. Available: http://tacodataset.org. [Accessed 9 August 2023].

[5] A. H. M. Z. A. Z. L.-C. C. Mark Sandler, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Google, 2019.

[6] S. SEKAR, "Kaggle - Waste Classification data," [Online]. Available: https://www.kaggle.com/datasets/techsash/waste-classification-data.

[7] K. Ambasht, "Understanding Precision, Recall, and Accuracy with COVID tests," *Towards Data Science,* 2020.

Appendix

The installation steps are as follows:

What you need:

1 Have python installed on your system: various methods, depending on your operating system.
2 Have Conda installed on your system: https://conda.io/projects/conda/en/latest/user-guide/install/index.html.

NOTE: cloning the environment provided in this submission will only work on a Linux system. Manual installation is required for other operating systems.

Darknet Setup (optional):

NOTE: this part of the project may or may function of your computer as you must correctly set the darknet library to your path.

You will need a Cuda compiler, Cudnn and a Cuda GPU driver. The Cuda drive should come with your GPU if you have correctly configured your machine after purchase. The Cudnn library requires you to create an account on nvidia. This is your choice; I will only provide the links steps below. **This is assuming you are using a Linux OS**

Before you start, you will need to create a conda environment. Refer to Yolov5 (below) steps 2 and 3 only. Once this is setup you may proceed.

1. Clone the repository: git clone https://github.com/AlexeyAB/darknet.git and refer to installation steps in the github repository.
2. Download cudnn https://developer.nvidia.com/cudnn
3. You will need a version of cuda: https://developer.nvidia.com/cuda-downloads
4. Inside the submission folder you will see a directory called darknet, go into it and move `yolov4-tiny-custom.cfg` into your `cfg` folder on your computer.
5. Inside the submission folder you will see a directory called darknet, go into it and move `yolov4-tiny-custom_final.weights` into your folder on your computer where darknet is.
6. Finally unzip the data directory in the onedrive and replace the data folder in your darknet with the one here.
    a. Enter the data directory
    b. Open test.txt, train.txt and data.yml. Replace all instances of `/home/flynn/482_project/darknet/data` with your path to the folder.
7. To run enter: `sudo ./darknet detector demo data/taco.data cfg/yolov4-tiny-custom.cfg yolov4-tiny-custom_final.weights -c 0`

In the end your folder structure should look like this:

```
|-- darknet
|    | yolov4-tiny-custom_final.weights
|   |-- data (this is the one provided in the Onedrive)
|   |-- cfg (contains yolov4-tiny-custom.cfg )
|   | ... remaining files
```

Note that the command: `sudo ./darknet detector demo data/taco.data cfg/yolov4-tiny-custom.cfg yolov4-tiny-custom_final.weights -c 0` must be executed in `| -- darknet` and not in any subfolders.

Unity GUI (Required):

To run the GUI, unzip the sorter_gui.zip, and open the folder in Unity. Press the play button and the app will wait for a socket connection from Python, described in the next section.

Yolov5 (Required):

You will need a **Linux** operating system do the following:

1. Navigate inside the `yolov5.zip` by unzipping the file
2. You will see a file called `yolov5_env.txt` inside the yolov5 folder. Enter the following command: `conda env create --file yolov5_env.txt --name yolov5_env`
3. Then activate the environment you just created from the text file: `conda activate yolov5_env`.
4. Then enter `pip install –r requirements.txt`
5. Download tensorflow: `pip install tensorflow`
6. Now you can run python `classifier_yolov5.py` OR python `unity_classifier.py`. If you decide to run these, you will need to adjust the path inside these scripts.
   a. **For only yolov5**, open the file named `classifier_yolov5.py`. Go to line 30 and replace with your path. For example, `weights = "/home/thisUser/thisFolder/yolov5/runs/train/exp23/weights/best.pt"`

      Navigate to the yolov5 folder and type: `python classifier_yolov5.py`

   b. **For the integrated system**, *you must first open unity* (refer to the unity steps before proceeding). Open the file named `unity_classifier.py`. Go to line 26 and replace it with the correct path. For example, `weights = "/home/thisUser/thisFolder/yolov5/runs/train/exp23/weights/best.pt”`. Do this again on line 32. For example, `model_classifier = tf.keras.models.load_model('/home/thisUser/482_thisFolder/yolov5/my_model.h5')`. You will also need to change line 47 to your computers IP address. You can find your IP address by typing `hostname -I`.

      To run navigate to the yolov5 folder and type: `python unity_yolov5.py`

If you encounter any problems, refer to the GitHub page of yolov5: https://github.com/ultralytics/yolov5