Application of Object Recognition for Plastic Waste Detection and Classification Using YOLOv3

I Wayan Raka Ardana
Electrical Engineering Department
Politeknik Negeri Bali, Indonesia
rakawyn@pnb.ac.id

Ida Bagus Irawan Purnama
Electrical Engineering Department
Politeknik Negeri Bali, Indonesia
ida.purnama@pnb.ac.id

I Made Sumerta Yasa
Electrical Engineering Department
Politeknik Negeri Bali, Indonesia
sumertayasa61@pnb.ac.id

2020 International Conference on Applied Science and Technology (iCAST) | 978-1-7281-9567-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/iCAST51016.2020.9557735

Abstract— **Object recognition is a computer vision technique to detect the semantic of objects either in digital images or videos then to identify those objects into a particular class. This intelligent technique can be used for various applications. In this study, object recognition is implemented for real-time plastic waste detection and classification using YOLOv3. Six macro plastic waste classes are proposed, namely plastic bag, plastic bottle, crushed bottle, cup, cartoon, and straw. These six categories are usually among the top of our daily plastic waste. This classification of plastics waste aims to make the sorting task more efficient both at home and recycling center. Using around 1858 images and 2000 iterations during dataset training, results show that the detection achieves a good confidence value for plastic bottle and cartoon class which is 85% and 75% consecutively. Meanwhile, straw achieves 65% and the others are between 30 and 40%. This means the algorithm can detect and classify the plastic waste correctly. However, the further review of images used in the dataset in terms of item variety, angle, lighting, and image resolution, as well as increase the iteration number during the training phase, are required to gain a higher confidence value.**

Keywords—plastic waste detection, computer vision, object recognition, YOLOv3, intelligent system

I. Introduction

Object recognition is one of the most prominent research tracks for computer vision [1]. How the object recognition work is first by feeding the computer images of the same type of object. Second, it finds common identifying features. Third, it uses these features to detect and identify the same objects in new images or real-time video. Object recognition with deep learning is being widely used not only in the industry right now [2] but also it is promising being implemented in the community level or even in the household scale. One promising application of object recognition is for waste management especially to detect and classify plastic waste. As the use of plastic is prevalent, plastic waste becomes a complex problem because it has a substantial impact not only on land and ocean but also on humans and animals. If it is not handled suitably, this potentially causes an unprecedented environmental crisis in the long term. Recognizing various plastics waste by appropriate classes will make the sorting task more efficient both at home and recycling center. However, the challenges could be the sufficiency of the training dataset that will be representative of the plastic waste model that will be applied to. This is due to the very diverse of plastic wastes in shapes, colors, and dimensions. Also, they have absorbance and reflectance characteristics in a certain spectrum.

Some works in the intelligent system have been done to classify garbage. The work of [3] classified waste into six classes consisting of glass, paper, metal, plastic, cardboard, and trash. Then, they created an image dataset that contains around 400-500 images for each class. Meanwhile, using this image dataset, [4] tried Hybrid Transfer Learning (HTL) for classification and Faster R-CNN to get region proposals for object detection. Then, [5] applied a machine learning approach using Convolutional Neural Network (CNN) as the detector with Support Vector Machine (SVM) as the classifier and achieved an accuracy of 87% on that dataset. These works need at least two algorithms for detection and classification.

YOLO (You Only Look Once) was initially introduced as the first object recognition model that integrated object classification and bounding box prediction into a single regression problem [2]. It is one of the popular algorithms in object recognition right now because it gains good accuracy along with being able to operate in real-time. YOLO has been used for various purposes of detection ranging from remote sensing [6], vehicle and traffic flow [7-9], cyclists [10] to fruits detection [11]. Different from the sliding window technique, the YOLO algorithm "seen just once" at the image then requires only one forward propagation pass through the network for making predictions. After non-max suppression, it gives the class name of the recognized object along with the bounding boxes around them and their level of confidence. These bounding boxes are not as random or arbitrary rectangles, but as an offset from one of the preconfigured bounding boxes called anchor boxes. In YOLOv3, it uses a package of dimensions to generate these anchor frames.

From the aforementioned context, this study aims to make a custom plastic waste dataset and implement object recognition using YOLOv3 in real-time to detect the plastic object, then to identify them into the correct classes. In this case, six classes of macro plastics are proposed, namely plastic bag, plastic bottle, crushed bottle, cup, cartoon, and straw.

II. YOLOv3 Theory

A. Bounding Boxes and Anchor Box

The use of bounding boxes for object detection makes only one object that can be identified by a grid. Therefore, for detecting more than one object anchor box is used. In YOLO, anchor boxes are used to predict bounding boxes in different sizes where the midpoint is placed in a similar cell, Fig. 1.

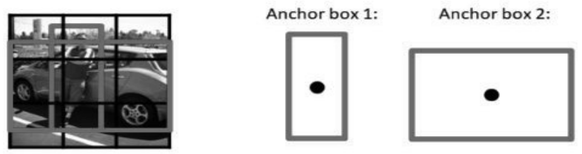


Fig. 1. Grid cells and anchor box

Consider the above figure, in that both the human and the car's midpoint come under a similar grid cell. The red color grid cells represent the two anchor boxes of those objects [1]. Any number of anchor boxes can be used for one image to detect multiple objects. Changing the anchor boxes number leads to a change in the length of ground truth and prediction array. In that figure, two anchor boxes have been taken.

B. YOLO Detection Process

The main concept of the YOLO framework is to split the entire input image into grid cells with S x S dimensions. Then, it makes detections in each grid cell by predicting B bounding boxes along with the confidence level of these boxes [11]. Here, confidence indicates the existence of an object in the grid cell. If it exists, the prediction of object and IoU (Intersection over Union) of the Ground Truth (GT) are used to calculate the confidence value as formulated in:

$$Confidence = Pr(Object) \text{ x } IoU(GT,pred) \quad (1)$$

where $Pr(Object)$ E $[0,1]$ and IoU is the ratio of intersection of two boxes to the union of the boxes as shown in Fig. 2 [1].

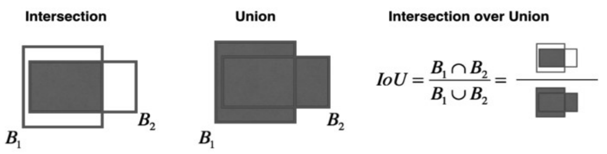


Fig. 2. Intersection over Union (IoU)

Besides the probability of the object *(p)* and bounding box specifying object location which consists of a center *(x,y)* and high width of the box *(h,w)*, each grid cell also predicts *C* class probabilities corresponding to a class of an object. So, there will be 5+C descriptors for each cell as shown in Fig 3.

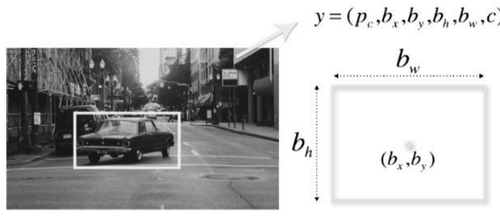


Fig. 3. Predicted values (descriptors) for each cell.

Now, each cell has a responsibility to predict a few different things. Firstly, it is responsible for predicting some number of bounding boxes and confidence value for every single bounding box. If there is no object in some grid cells, the confidence value will be very low for that cell. When all of these predictions are pictured, a map of all the objects with a bunch of boxes that is ranked by their confidence value will be gained. Secondly, each cell is responsible for predicting class probabilities. But, this does not mean that some grid cell contains some object, this is just a probability. Hence, if a grid cell predicts a dog, it does not mean that there is a dog, it just means that if there is an object then that object is a dog.

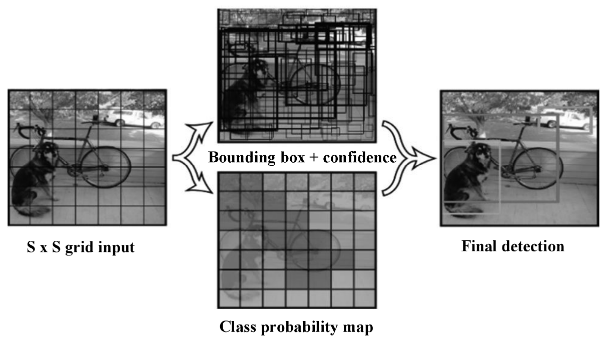


Fig. 4. YOLO detection process

III. Material and Method

The experiment method in this study is conducted in four steps as depicted in Fig. 5.

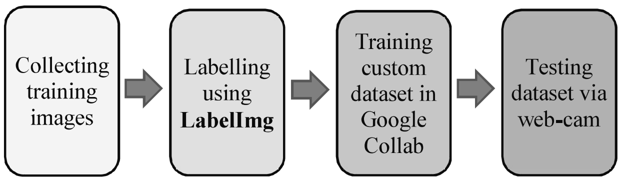

Collecting training images
Labelling using LabelImg
Training custom dataset in Google Collab
Testing dataset via web-cam

Fig. 5. The four steps of experiment method.

The first step is collecting images for training data, which is what the model will use to find identifying features of objects in prediction. We collected random images of plastic waste from the internet. Afterward, they are classified into six classes, namely plastic bag, plastic bottle, crushed bottle, cup, cartoon, and straw, Fig. 6. In total, 1858 images of plastic waste are collected. Each category consists of 492, 245, 297, 368, 131, 325 images of the plastic bag, plastic bottle, crushed bottle, cup, cartoon, and straw consecutively.

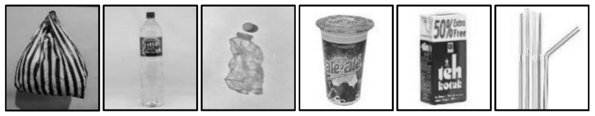


Fig. 6. The proposed six classes of the plastic waste

The second step is labeling the images. To train the object detector, it needs to supervise its learning with bounding box annotations. We have to draw a box around each object that we would like the detector to focus on. Then, give every box a label with the object class that we want the detector to predict. This study used the **LabelImg** annotation tool to draw the bounding boxes around plastic objects on the images and label them as their class. A label file will be created for each image where both label files and their respective image files are kept together, Fig. 7. In drawing of bounding boxes, some useful things should be followed such as label around the object in question, label obstructed objects entirely, and avoid too much space around the object in question. It is best to draw bounding boxes that include the entire objects, even if there is a small amount of space between the bounding box and the object. In other words, do not cut out any of the underlying objects with the bounding boxes.

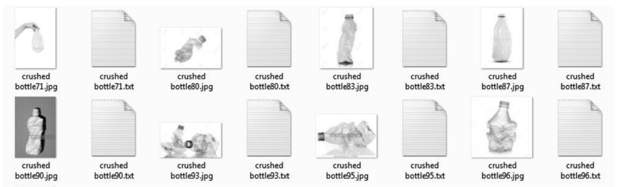


Fig. 7. Example of images and their respective label files for crushed bottle

The third step is to train our custom dataset for some iteration (batches) to get weight results. This specific model is a one-shot learner. This means each image only gets through the network once before making a prediction. This also allows the architecture to be good enough, viewing up to a certain fps (frame per second) in predicting with video feeds. YOLOv3 splits an image into subcomponents. Then, after running convolutions on each of the subcomponents, it pools back to make a prediction. To power our model's computation, Google Colab is used. It provides free GPU compute resources (up to 24 hours with your browser open).

The last step is to test the model based on the flowchart below. Then, the following Python script is used by feeding the web-cam new objects of plastic that are not in the dataset. The experiment is conducted for a single object and multi objects. Whether it could detect and identify the plastic object class correctly is then examined.

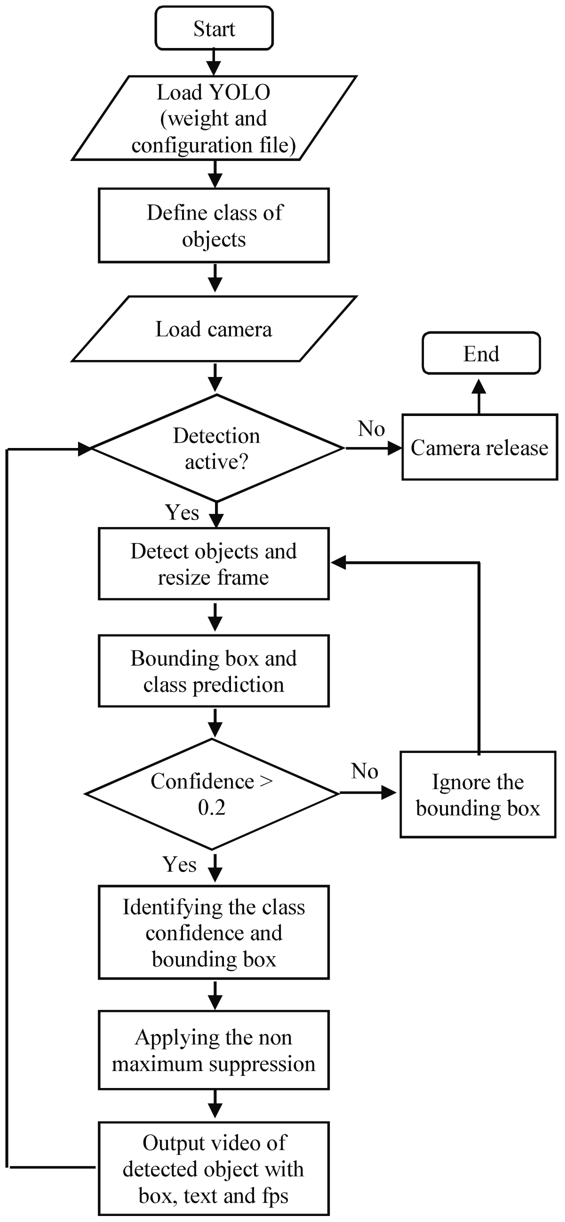

Start
Load YOLO (weight and configuration file)
Define class of objects
Load camera
End
Detection active?
No
Camera release
Yes
Detect objects and resize frame
Bounding box and class prediction
Confidence > 0.2
No
Ignore the bounding box
Yes
Identifying the class confidence and bounding box
Applying the non maximum suppression
Output video of detected object with box, text and fps

Fig. 8. Flowchart of testing YOLOv3 model

The Python script derived from the flowchart uses **dnn module** of OpenCV version 2. It has a deep learning library that can work with YOLOv3. To run the algorithm, it needs two files: (1) **Weight file** which is the trained model, the core of the algorithm to detect the objects, (2) **Cfg file** which is the configuration file, where there are all the settings of the algorithm. Meanwhile, classes contain the name of the objects that the algorithm can detect. The full image on the network cannot be used right away, but first, it needs to convert into **blob**. Here, blob is used to extract features from the image and to resize them where we use sizes 320×320. It is small so less accuracy but better speed. The threshold confidence is set to 0.2. When performing the detection, it happens that for the same object it has more boxes, so a function called Non-max Suppression is used to remove this "noise". Finally, box, label, confidence, and fps are shown on the screen.

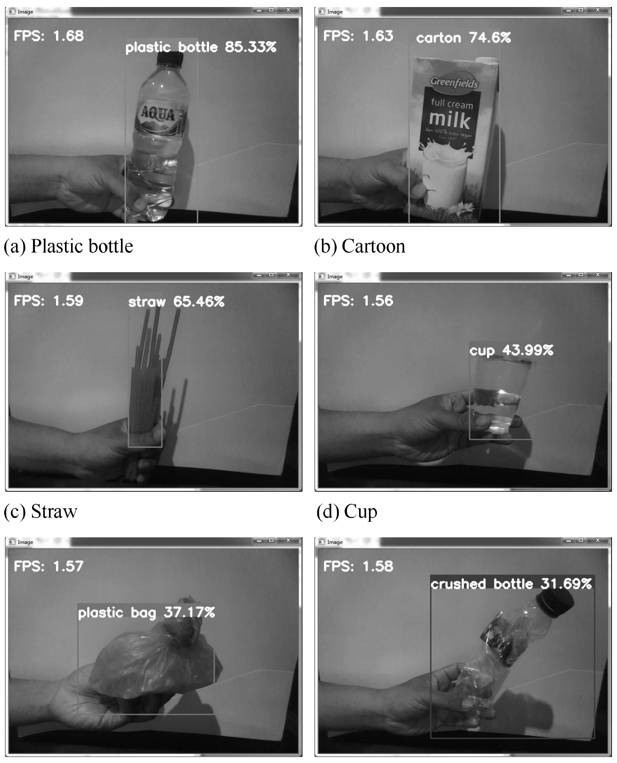
```
import cv2
import numpy as np
import time

# Loading Yolo
net = cv2.dnn.readNet("yolov3_training_last.weights", "yolov3_custom.cfg")
classes = ['plastic bag','plastic bottle', 'straw', ' carton', 'cup', 'crushed bottle']
layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Loading camera
cap = cv2.VideoCapture(0)
font = cv2.FONT_HERSHEY_SIMPLEX
starting_time = time.time()
frame_id = 0

while True:
   _, frame = cap.read()
   frame_id += 1
   height, width, channels = frame.shape

   # Detecting objects
   blob = cv2.dnn.blobFromImage(frame, 0.00392, (320, 320), (0, 0, 0), True,
   crop=False)
   net.setInput(blob)
   outs = net.forward(output_layers)

   # Showing informations on the screen
   class_ids = []
   confidences = []
   boxes = []

   for out in outs:
      for detection in out:
         scores = detection[5:]
         class_id = np.argmax(scores)
         confidence = scores[class_id]

         if confidence > 0.2:
            # Objects are detected
            midpoint_x = int(detection[0] * width)
            midpoint_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            # Rectangle coordinates are created
            x = int(midpoint_x - w / 2)
            y = int(midpoint_y - h / 2)
            boxes.append([x, y, w, h])
            confidence1 = confidence*100
            confidences.append(float(confidence1))
            class_ids.append(class_id)

   indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.4, 0.3)

   for i in range(len(boxes)):
      if i in indexes:
         x, y, w, h = boxes[i]
         label = str(classes[class_ids[i]])
         confidence = confidences[i]
         color = colors[class_ids[i]]
         cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
         cv2.rectangle(frame, (x, y), (x + w, y + 30), color, -1)
         cv2.putText(frame, label + " " + str(round(confidence, 2)) +"%", (x, y + 30),
         font, 1, (255,255,255), 3)

   elapsed_time = time.time() - starting_time
   fps = frame_id / elapsed_time
   cv2.putText(frame,"FPS: "+str(round(fps, 2)), (10, 50), font,1, (255, 255, 255),3)
   cv2.imshow("Image", frame)
   key = cv2.waitKey(1)
   if key == 27: #This is ESC keyboard
      break

cap.release()
cv2.destroyAllWindows()
```

IV. Results and Discussion

This section presents the real-time experiment results using a CPU core i7 with its camera. It also discusses their significances. The further implementation of this algorithm for practical use using embedded devices is presented too.

A. Real-Time Detection Results

By holding the object in front of the camera, Fig. 9 shows the real-time detection and identification results for each class of plastic waste. The aim of holding the object instead of placing them steady on the table is to get the moving object effect so that the bounding box can follow the object shifting.



(a) Plastic bottle (b) Cartoon

(c) Straw (d) Cup

(e) Plastic bag (e) Crushed bottle

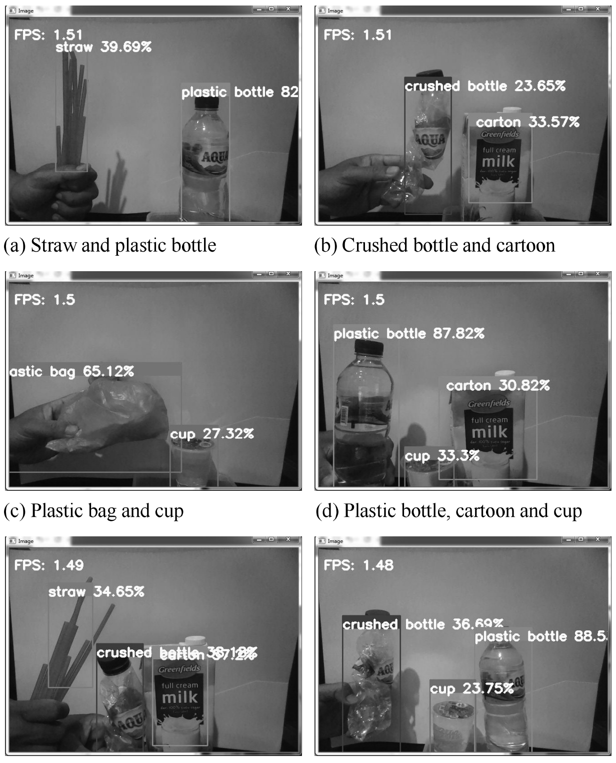
Fig. 9. Real-time detection results of single object

The plastic bottle and cartoon are detected with high confidence of 85.33% and 74.6% successively. Then, straw is detected with the middle level of confidence which is 65.46% and the cup, plastic bag, and crushed bottle are between 30% and 40%. In this case, the hand which holds the objects is not detected because it is not in the dataset. Table I below shows the confidence values of single and multi-object detection.

TABLE I. Detection Results

Plastic objects	**Confidence Detection**	
	In Single Object Detection	***In Multi-object Detection***
Plastic Bottle	85.3%	82.1%, 87.8%, 88.5%
Cartoon	74.6%	33.5%, 30.8%, 37.2%
Straw	65.4%	39.6%, 34.6%
Cup	43.9%	27.3%, 33.3%, 23.7%
Plastic Bag	37.1%	65.12%
Crushed Bottle	31.6%	23.6%, 38.1%, 36.6%

In multi-object detection, only the plastic bottle can maintain a high confidence value, relatively above 80%. This means that the training dataset for the plastic bottle is enough to keep its confidence level. Meanwhile, the cartoon drops to around 30%. Other objects are still in low confidence value. Fig. 10 shows the image detection for all multi-objects. Here, the detection of the class is still correct but the confidence other than the plastic bottle is still low. This indicates that different angles and positions of the object may affect the confidence so that retrain the dataset with more objects is needed. FPS (Frame per Second) shows the frame rate which is the speed at which those images are shown on the screen.



(a) Straw and plastic bottle (b) Crushed bottle and cartoon

(c) Plastic bag and cup (d) Plastic bottle, cartoon and cup

(e) Straw, crushed bottle and cartoon (f) Crushed, plastic bottle, and cup

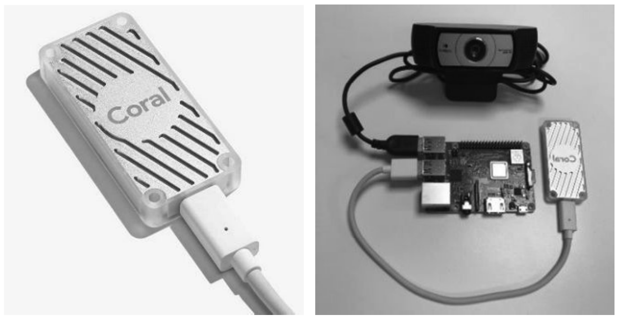
Fig. 10. Real-time detection results of multi objects

B. Results Discussion and Practical Application

The experiment results show that all objects can be identified and classified correctly, but only two classes achieve a good level of confidence which is higher than 70% in single object detection and only one in multi objects detection. The position of the object during detection has a significant impact especially for objects with irregular forms such as plastic bags and crushed bottles. If objects are visually dissimilar, the model had trouble in finding common identifying features. This indicates that not all angle positions of images are available in the dataset. Therefore, reexamining images used in the dataset could be useful.

As the code here does not use the Graphics Processing Unit (GPU) capabilities of the system for image processing, the required to process the frames only by CPU is extensive. This results in a very low FPS and displays latency. To process a frame and also display the bounding box over the detected objects, the trained model requires about 0.6 seconds. The detection performance can be significantly improved by making use of the GPU in the respective system [1]. The detection fails for the objects that are far from the camera view due to the images in the training dataset had the objects to be detected in focus and thus had more object frame to the size of the image ratio [1]. The model performs better in the environment with optimum lighting conditions. Also, cleaning and augmenting image data in terms of angle and image resolution could improve the ultimate model's performance. Retraining the dataset with higher iteration is also useful.

For practical use, this algorithm can be deployed into embedded devices so it can then be combined with mechanical devices for a plastic waste management application. As the algorithm is written in Python, Raspberry Pi 4 with 4 GB can be used. However, this YOLOv3 may need higher resources to run so that Raspberry Pi 4 with a webcam can be combined with the aid of Google Coral USB accelerator or Intel Movidius NCS to make it faster, Fig. 11. But, if near real-time detection cannot be achieved with this plan, Tiny-YOLO can be the alternative.



(a) Coral USB Accelerator (b) Raspberry Pi, Webcam and Coral

Fig. 11. Embedded devices for object detection.

After the algorithm running in the embedded device, it can be further used for a real application in intelligent waste segregator and sorter. It may need mechanical devices and structures such as motor, conveyor, and selector so that it can automatically throw the plastic waste to the specific bin or thrash container based on their classes as soon as they are recognized. Thereby, this will make waste management more efficient.

V. CONCLUSION

This study focusses on plastic waste detection and classification using the object recognition technique. Dataset is built using 1858 images where six classes of macro plastic waste are proposed, namely plastic bag, plastic bottle, crushed bottle, cup, cartoon, and straw. These six classes are usually among the top of our daily plastic waste. Dataset was trained by 2000 iteration. The real-time object detection experiment using YOLOv3 has been conducted to identify plastic waste from each category using a built-in camera in a CPU Core i7. Results show that single object detection achieves a good confidence value for plastic bottle and cartoon category which is 85% and 75% consecutively. Meanwhile, straw achieves 65% and the others are between 30 and 40%. This means the algorithm can detect and identify the plastic waste correctly. However, the further review of images used in the dataset in terms of item variety, angle, lighting, and image resolution, as well as increase the iteration number during the training phase, are required to gain a higher confidence value. This algorithm also can be deployed into embedded devices for practical use with the aid of an accelerator to make it faster.

ACKNOWLEDGMENT

Authors wish to acknowledge that this work is fully supported and funded by the Centre of Research and Community Services, Politeknik Negeri Bali, Indonesia, via Grant of DIPA PNB 2020.

REFERENCES

[1] O. Masurekar, O. Jadhav, P. Kulkarni, and S. Patil, "Real-Time Object Detection Using YOLOv3," International Research Journal of Engineering and Technology (IRJET), vol. 07, issue 03, pp. 3764–3768, 2020.

[2] A. Vidyavani, K. Dheeraj, M. Rama Mohan Reddy, and KH. Naveen Kumar, "Object Detection Method Based on YOLOv3 using Deep Learning Networks," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 9, issue 1, pp. 1414–1417, 2019.

[3] M. Yang and G. Thung, "Classification of Trash for Recyclability Status," Stanford University, 2016.

[4] H. N. Kulkarni and N. K. S. Raman, "Waste Object Detection and Classification," CS230, Stanford University, 2018.

[5] O. Adedeji and Z. Wang, "Intelligent Waste Classification System Using Deep Learning Convolutional Neural Network," Procedia Manufacturing, 35, pp.607-612, 2019.

[6] W. Zhihuan, Xiangning C, Y. Gao, and Yuntao. Li, "Rapid Target Detection in High-Resolution Remote Sensing Images using Yolo Model," ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLII–3, pp. 1915–1920, Apr. 2018.

[7] S. S. A. Rajjak and A. K. Kureshi, "Object Detection and Tracking using YOLO v3 Framework for Increased Resolution Video," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 09, issue 06, pp. 118–125, 2020.

[8] J. Sang, Z. Wu, P. Guo, H. Hu, H. Xiang, Q. Zhang, and B. Cai, "An Improved YOLOv2 for Vehicle Detection," Sensors, vol. 18, no. 12, p. 4272, Dec. 2018.

[9] Y. Q. Huang, J. C. Zheng, S. D. Sun, C. F Yang, and J. Liu, "Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections," Applied Science-MDPI, vol. 10, 3079, pp. 1–15, 2020.

[10] C. Liu, Y. Guo, S. Li, and F. Chang, "ACF Based Region Proposal Extraction for YOLOv3 Network Towards High-Performance Cyclist Detection in High-Resolution Images," Sensors, vol. 19, no. 12, p. 2671, Jun. 2019.

[11] G. Liu, J. C. Nouaze, P.L.T. Mbouembe, and J. H. Kim, "YOLO-Tomato: A Robust Algorithm for Tomato Detection Based on YOLOv3," Sensors-MDPI, vol. 20, 2145, pp. 1-20, 2020.