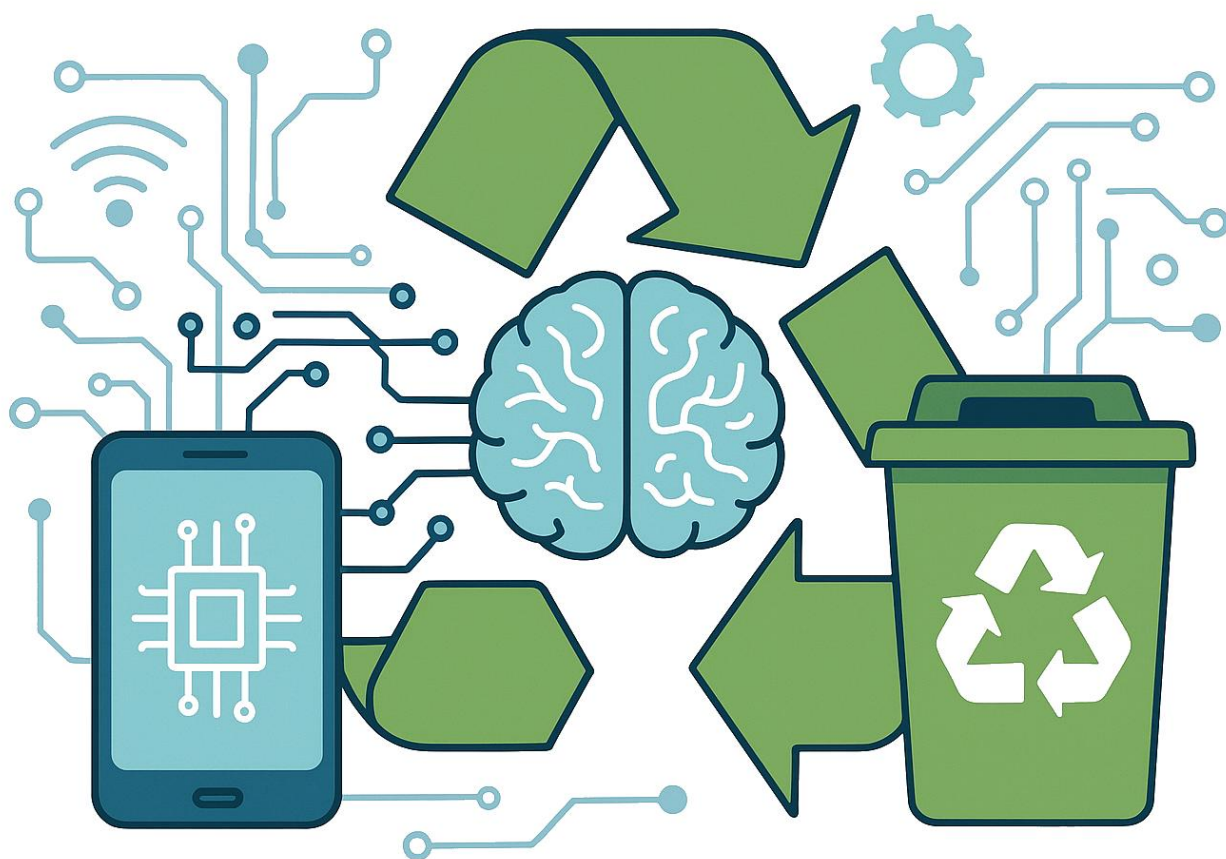


中華民國第 65 屆中小學科學展覽會

作品說明書



科 別：電腦與資訊學科

組 別：高級中等學校組

作品名稱：利用深度學習實作資源回收系統之設計

關鍵詞：深度學習、自動資源回收、物聯網

編號：

摘要

本研究以深度學習 YOLO11 為核心，開發一套智慧資源回收辨識系統，部署於 Raspberry Pi 5 平台上，透過模型優化與自動化判斷，有效降低回收分類錯誤與人力負擔。系統具備即時影像擷取、回收物分類、資料標註與上傳等功能，並能自動將分類結果與影像同步儲存至 Google Drive，作為後續增量訓練資料，提升模型準確率。

同時，結合物聯網架構與 LINE Notify 實現異常警報與即時訊息推播，協助管理者即時掌握設備狀況與分類情形，並產製雲端圖表進行視覺化分析。實驗結果顯示，系統辨識置信度穩定達 0.7 以上，平均運行速度為 15~30 秒，效率良好，成功實現回收分類之智慧化、自動化與高效能，對提升回收準確率與推動智慧城市資源管理具實質應用價值。

壹、前言

一、研究動機

隨著科技不斷進步，生活水準提升之餘，人民對環保的意識也隨之提高。儘管政府意識到人民每天產生的垃圾數量爆增，並著手處理垃圾及資源回收問題，但台灣民眾的一般廢棄物回收率仍然只有 50~60%。從日常生活中，可觀察到民眾對政府推動資源回收的分類規則感到困惑，本研究認為這是導致廢棄物回收率無法進一步提升的主因。因此，本研究希望透過深度學習，以智慧化自動辨識廢棄物，可區分為垃圾或資源回收類，使廢棄物處理變得更容易，進而提高資源回收率，並為地球環保盡一份心力。

本研究認同聯合國永續發展目標（SDGs）（The Global Goals. (2025) [1]），例如：

SDG10（減少不平等）：本研究的回收系統除了螢幕和燈光外，還有聲音提示，能向視障者和老年人等族群，提供使用智慧回收機的機會，促進科技的無障礙應用。

SDG11（永續城市與社區）：正確的垃圾分類能夠減少垃圾掩埋的需求，提高回收率，進而減少每人對環境的負面影響。

SDG12（負責任的消費與生產）：視覺辨識模型能精確辨認並分類廢棄物，使更多的自然資源繼續被循環利用，這不僅能提升回收效率，也有助於建立更完善的循環經濟模式，促進永續消費與生產。

SDG13（完備減緩調適行動，以因應氣候變遷及其影響）：準確的回收能有效降低可回收物被錯誤焚燒的機率，進而減少二氧化碳與其他溫室氣體的排放，藉此來減緩氣候變遷。

此外，本研究在產品中新增螢幕、燈光及聲音等感測裝置，以進行人機互動的實用及娛樂性，期待透過這樣的方式，提高一般民眾對利用深度學習的 AI 智慧產品好奇心與興趣，藉此理解與支持對廢棄物處理的研究產品，共同為更美好地球的目標邁進。

二、研究目的

本研究透過分析深度學習 YOLO (You Only Look Once) 在視覺辨識的最新進展，理解其概念與資訊，在樹莓派 5 (Raspberry Pi 5) 實作深度學習的綜合應用，期待完成資源回收系統之設計，最後並提出評測與建議。除深度學習外，研究中亦綜合不同的元素，為研究產品創造新的功能，例如：應用感應技術，讓資源回收裝置能自動開啟，並進行回收流程。此外，本研究也有 AIoT「人工物聯網」(Artificial Intelligence of Things) 功能，透過 LINE Notify 即時接收產品的訊息以及結合 Google Drive 收集與整理數據上傳到雲端。總結來說，本研究透過人工智慧的深度學習，創造一個更輕鬆、更有效，且低成本的產品，能快速解決廢棄物是垃圾或資源回收問題，協助民眾做環保，促進地球更美好。以下是本研究的目的：

- (一) 分析最新 YOLO 的應用。
- (二) 如何使用 YOLO 的標記格式完成 YOLO 的模型訓練。
- (三) 應用 YOLO 的訓練成果部署在 Raspberry Pi 5。
- (四) 可即時、快速、辨識資源回收分類（一般垃圾、紙容器、塑膠及金屬）。
- (五) 自動將檢測的詳細數據整理並上傳到雲端。

三、文獻回顧

(一) 資源回收機的現況

目前市面上有兩種主要的資源回收機系統，分別由 7-11 超商和新北市政府所推廣。7-11 的「iCIRCLE 高效智慧回收機」目前只接受「寶特瓶」和「電池」兩種回收分類，它是利用 AI 光學辨識技術回收寶特瓶，因此當寶特瓶上方有較複雜的包裝紙，寶特瓶裡面仍有水，又或者是寶特瓶瓶身的顏色較深，都可能導致機器辨識問題，因而將回收物"退貨"，這台機器的特點在於具有碾碎功能，可將寶特瓶碾成碎片，節省體積。（統奕，2025 [2]）

「新北市資源循環教育基地」以 AI 技術用於精細化的回收物分選，也引入「光選機」和「磁選機」在內的自動化系統，能夠針對不同材質（如塑膠、金屬）進行快速且精確的分選，同時使用智慧回收機器人，用「機械手臂」和「光學辨識系統」回收各種分類，總共可回收 56 種資源。只不過因為回收的種類多，且大型機械眾多與複雜，因此就算比傳統回收

作業的使用土地較少，但本身回收機還是佔有一定的規模，而且造價也不便宜。

（二）YOLO 發展現況

YOLO (You Only Look Once) 是一種基於深度學習的即時物件偵測演算法，能夠在單次神經網路運算內，同時執行物件分類 (Classification) 與定位 (Localization)。YOLO 相較於傳統的區域提案 (Region Proposal) 方法，YOLO 透過將影像劃分為網格，讓每個網格負責預測邊界框及物件類別，使其具備高速、準確且適合即時應用的特點。

YOLO 的單鏡頭檢測方法，將圖片劃分為各個網格，且直接為每個網格單元預測邊界框和類別概率，使得 YOLO 更快速且高效，相較於 Faster R-CNN 等兩階段模型，實現了顯著的速度提升。

傳統方法 (如 Faster R-CNN) 先找出可能的物件區域，再逐一分類，因此運行速度較慢。YOLO 直接將圖片劃分為網格，每個網格獨立負責物件偵測，減少冗餘計算，因此速度大幅提升。(Darwin. 2022 [3])

傳統 CNN 主要依靠層層堆疊的卷積與池化操作來提取特徵，YOLO 則採用不同尺度的特徵圖來提升辨識能力。這樣不但可以兼顧大範圍的全局資訊 (來自 Backbone)，也能夠保留局部細節 (來自 Neck 的特徵融合)，提升模型的準確度和泛化能力。

YOLO 系列持續的在進化，2024 至 2025 的發展趨勢，YOLO 有一系列的更新。以 YOLOv8 為例，由 Ultralytics 開發，YOLOv8 是一個相當多功能的模型，不僅支援物件偵測，還包含影像分割和姿勢估計等任務。它被設計為速度、準確性和靈活性兼具的解決方案。YOLOv8 的特色在於它更易於使用，並在各種應用中都有良好的適應性。(Ultralytics. 2025 [4])

YOLO11 則是 2024 年 9 月推出的新模型，與前代產品相比，此版本有多項重大改進。

- 增強特徵提取：YOLO11 採用改進的指標和框架，增強了特徵提取，從而實現更精確的目標檢測。
- 解決方案的效率和速度：經過改進的架構設計和訓練流程可提供更快的解決方案，同時保持準確性和效能之間的平衡。
- 參數更少，精度更高：與 YOLOv8m 相比，YOLO11m 在 COCO 資料集上實現了更高的平均精確度(mAP)，而參數數量卻配置了 22%，這使得在不影響精確度的前提下提高了計算效率。
- 跨環境適應性：YOLO11 可在各種安裝程式中部署，包括邊緣設備、雲端平台和支援

NVIDIA GPU 的系統。

- 支援任務範圍廣泛：YOLO11 支援多種電腦視覺任務，如物件偵測、實例分割、分類、姿態估計和定向物件偵測（OBB）。

為確保模型運行速度與準確率兼具，本研究比較了 YOLOv5、YOLOv8、YOLOv11，並選擇 YOLOv11 的版本進行訓練，因為它在速度與精準度之間取得了較好的平衡，適合即時垃圾分類的應用場景。

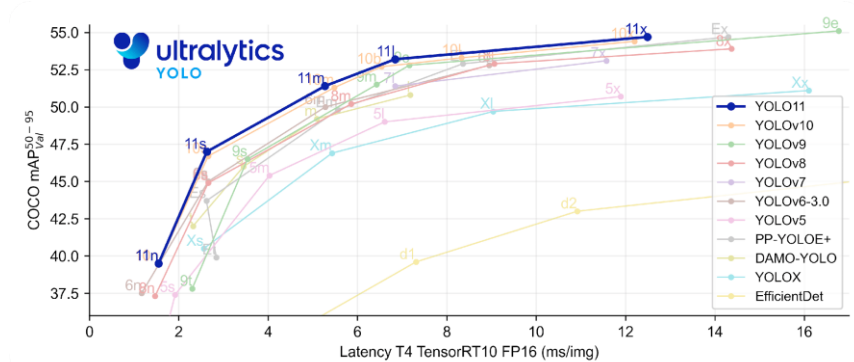


圖 1 YOLO11 與其它版本比較（資料來源：Darwin. 2022 [2]）

本研究經文獻分析後，決定採用 YOLO11 模型作視覺核心技術，由於 YOLO11 還可細分為 YOLO11n、YOLO11s、YOLO11m、YOLO11l 和 YOLO11x 這五個版本，版本的主要差異是模型精度和運行速度，模型精度和運行速度又被參數數量、層數、通道數、計算量（FLOPs）以及 backbone 結構所決定，在這之中 YOLO11n 為最小型的模型，參數數量少、運行次數少、運行速度最快，但準確度是最低的。而 YOLO11s 的運行速度相較 YOLO11n 較慢，但準確率卻高出不少，如下表所示：

表 1 YOLO11 在 COCO 訓練的範例

模型	尺寸 (像素)	mAPval 50-95	速 度 CPU ONNX（毫秒）	速 度 T4TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

資料來源：Ultralytics. (2025). [4]

貳、研究設備及器材

本研究的設備及器材，如表 2、表 3 所示：

表 2 研究設備表

項目	規格
桌上型電腦	1. CPU: I5-12600KF 3.7Ghz 2. RAM: 32GB DDR5-5600 3. SSD: 1TB 4. GPU: Nvidia GeForce RTX4060
作業系統	Windows 11
辨識環境	YOLO11s
樹梅派主機板	Raspberry Pi 5B 8GB +64GB
協助影像標記軟體	Roboflow
3D 圖繪製軟體	Tinkercad

資料來源：研究者自行製作

表 3 研究器材表

主要項目	規格	次要項目	規格
相機	C920Pro	樹梅派原廠電源線	Raspberry Pi 5 USB-C27W
超音波感測器	HC-SR04	按鈕	Kailh 海軍藍軸
伺服馬達	MG90S	步進馬達驅動器	DA2D24BP
步進馬達	Oriental motor	繼電器	二路 5V
喇叭	5W 喇叭揚聲器	電源供應器	150W/24V
LED 燈	5mm	麵包板	800 孔 (165*55mm)
電磁鎖	24V	杜邦線	10cm、20cm、30cm
滑軌	RXP45	電阻	1k Ω 、2k Ω
顯示螢幕	OLED		

資料來源：研究者自行製作

參、研究過程與方法

基於前述 YOLO 技術的發展與分析現有資源回收的現況，本研究決定採用最新 YOLO11s 模型，因其能快速與正確物件分類。本組使用 Ultralytics2024 開源的 YOLO11s 模型，以 Raspberry Pi 5 作運行核心，處理視覺辨識與操控資源回收任務。

為使上述任務發展順利，可成功開發研究產品，本研究首先執行 YOLO11s 物件分類模型訓練，得到最佳辨識模型後，研究者以 Python 設計演算法，整合應用完成資訊回收之設計。

本研究可分成三個研究方向：包括模型訓練、系統演算法及系統整合測試，茲分述如下：

一、模型訓練

本研究採用模型訓練的架構如圖 2 所示。

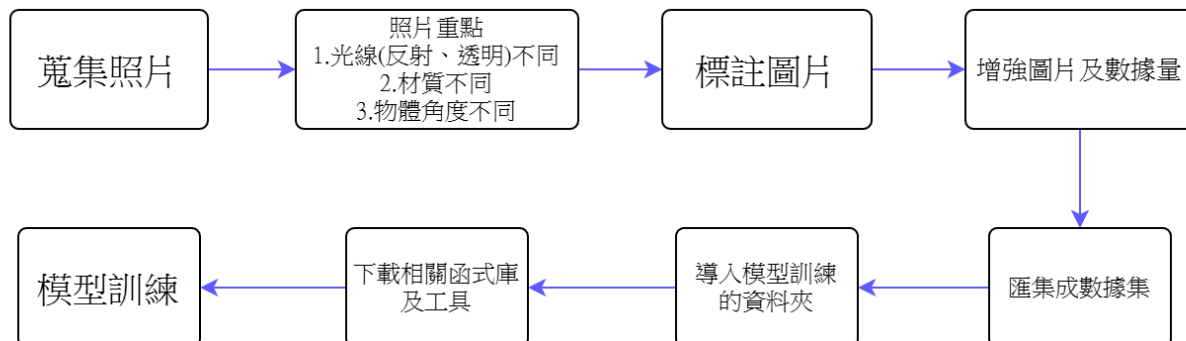




圖 2 視覺辨識模型訓練架構（資料來源：研究者自行製作）

本研究為取得數量大且圖片品質優的資料集，透過以下程序進行各項任務。

1. 收集圖片

由於本研究採用 YOLO11 訓練模型，這模型需要大量圖片作基礎，且其效能高度依賴於訓練資料集的品質，圖片不準確或不一致的標註（例如，錯誤的邊界框或類別標籤）會導致模型效能下降，且圖片不同的拍攝角度，或物體被遮擋或重疊，也會降低模型的辨識水準。所以研究者在圖片收集及圖片標記上特別著重以下重點，如下表所示。

表 4 YOLO 模型訓練之圖片影響程度及解決方法

圖像要素	影響	解決方法	例圖
光線	低光或逆光影響檢測置信度。	增強亮度、多種光照條件。	 圖 3-1 低光  圖 3-2 曝光

材質	反光、透明物體較難被識別。	增加不同材質數據或減少曝光度。	 <p>圖 3-3 反光</p>  <p>圖 3-4 透明</p>
角度	翻轉、側面角度可能影響識別，甚至無法被偵測到。	除了增加更多不同角度的物品，也可使用旋轉等增強功能。	 <p>圖 3-5 特殊角度</p>
遮擋	模型可能無法正確識別物品正確種類。	仍需要標記該回收物可見部分的特徵，以幫助模型學習如何處理遮擋情境。	 <p>圖 3-6 遮擋</p>
背景複雜	若背景過於複雜，可能會導致誤判物品種類，且置信度降低。	盡量在單一顏色背景下拍攝回收物。	 <p>圖 3-7 背景混雜</p>
形變	可能導致 YOLO 學習不到穩定特徵，影響置信度。	新增所有可能的形狀數據或透過拉伸等增強功能。	 <p>圖 3-8 形變</p>

資料來源：研究者自行製作

2. 標記物件

為了讓訓練模型能夠準確識別不同類別的垃圾，本研究透過數據管理平台 Roboflow 為所有圖片快速標註，中間進行數據增強和轉換數據集格式(NCnn)，並建立一個包含多種資源回收（塑膠、金屬、紙容器）的類別標籤數據集。匯出的數據集中，每張圖片都會有一個對

應的標籤文件，標記格式如表 5 所示：

表 5 圖片標記內容

class_id	x_center	y_center	width	height
檢測類型	物件框中心的 x 座標	物件框中心的 y 座標	物件框選寬度尺寸	物件框選高度及尺寸
		<div><div>檔案編輯檢視</div><div>2 0.5140625 0.515625 0.8296875 0.2328125 </div><div>class_id : 2 x_center : 0.5140625 y_center : 0.515625 width : 0.8296875 height : 0.2328125</div></div>		

圖 4 數據集圖片

資料來源：研究者自行製作

其中，class_id 代表回收物的類別，x_center 和 y_center 為物件中心座標，width 和 height 則代表物件的尺寸，所有數值皆為相對於影像大小的比例。透過標籤文件便可讓模型了解物體類型及其位置。





3. 數據增強

為了增加數據的多樣性，本研究透過旋轉、縮放以及改變亮度等方式進行數據增強，助於提升模型對不同光照、角度及背景變化的適應性，使其在真實應用中有更穩定的表現。透過此類數據增強功能不僅可增強辨識物件的能力，更能減少數據需求量，並縮減訓練時間。

4. 訓練 YOLO 模型

透過文獻分析，本研究擬採用最新 YOLO11 進行模型訓練，但 YOLO11 包括 5 種不同版本的模型，因此本研究先以測試集 168 張圖片數據，比較 YOLO11 各模型之間的檢測結果，以及是否準確辨識，其中 YOLO11x 因訓練時間過長，且檢測所需時間較長，故不列入討論。分析結果如下表所示：

表 6 YOLO11 各模型的辨識結果比較

模型 n	模型 s	模型 m	模型 l
			
圖 5-1 模型 n 結果	圖 5-2 模型 s 結果	圖 5-3 模型 m 結果	圖 5-4 模型 l 結果

資料來源：研究者自行製作

由上述實驗得知，在金屬方面，模型 s 和 m 的辨識能力較好，而 n 和 l 在這方面有辨識到兩種類型的錯誤。而在紙容器方面，只有模型 m 有辨識出特殊角度的紙箱。最後在塑膠方面，模型 n 和 l 在檢測塑膠瓶有較好的表現。在綜合考量下，本研究決定使用 YOLO11s 模型。

在 YOLO11s 模模型訓練前，須預先配置文件，用於指定數據集路徑、類別數和類別名稱，如下圖所示。

```
! data.yaml
C: > Users > SungPIPI > Desktop > yolo11 > ! data.yaml
1 train: ../train/images #train數據集檔案位置
2 val: ../valid/images #val數據集檔案位置
3 test: ../test/images #test數據集檔案位置
4
5 nc: 3 # 類別數
6 names: ['Metal', 'paper', 'plastic'] # 類別名稱
```

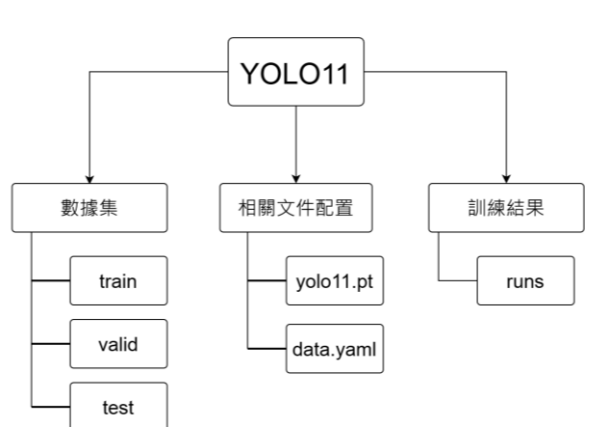


圖 6 模型訓練前配置規劃（資料來源：研究者自行製作）

為確保數據格式符合 YOLO 訓練要求，本研究使用 OpenCV 進行影像標準化，例如將所有圖片調整至 640x640 像素，並確保標記數據的完整性，避免標記錯誤導致模型訓練品質不佳。

5. 前置工具及函式庫

本研究分析 Ultralytics 關於 YOLO11 說明文件，下載相關函式庫，並作訓練前的 CUDA 環境設定，詳如下表所示：

表 7 YOLO11 訓練的工具與函式庫

函式及工具	說明
venv	Python 內建的虛擬環境管理工具，用來在不同專案中維護獨立的 Python 環境。
numpy	用於數值計算，提供多維陣列和高效的數學運算功能。
pandas	用於數據分析和處理的工具。
matplotlib	繪製數據圖表和可視化數據的工具，用於繪圖和資料視覺化。
opencv-python	用於電腦視覺處理，支援圖像和影片的讀取、處理、特徵檢測等，提供強大的圖像處理功能。
pytorch	深度學習框架，用於深度學習模型構建和訓練。
torchvision	提供 PyTorch 的計算機視覺工具，包括圖像數據集處理、預訓練模型等
ultralytics	Ultralytics 開發的函式庫，提供 YOLO（You Only Look Once）物件偵測模型的工具包。
CUDA	NVIDIA 提供的一種 GPU 平行運算架構，用來加速科學計算、深度學習、電腦視覺等應用。透過自有的 NVIDIA 顯示卡並安裝 CUDA，PyTorch 會自動使用 GPU 加速。

資料來源：研究者自行製作

6. 模型訓練

在模型訓練階段，本研究將數據集分成訓練集與驗證集兩部分，其中訓練集有 3720 筆標記後圖片，驗證集有 330 筆圖片。隨後本研究在桌上型電腦 (GPU Nvidia GeForce RTX4060) 上訓練 YOLO11s 模型，並設定適當的參數，如表 7 所示：

表 8 模型訓練的參數

參數設定	說明
Batch size = 16	一次處理 16 張圖片，加速訓練
Learning rate = 0.001	控制學習步伐，避免收斂過快或過慢
Epochs = 500	訓練 500 次，確保模型達到最佳效果

資料來源：研究者自行製作

同時，本研究使用 Adam（Adaptive Moment Estimation，適應性動量估計，又稱優化器）來提升模型學習效率，使其更快收斂到最佳狀態。

7. 訓練監控與驗證

在訓練過程中，本研究透過損失函數（Loss function）監控模型的學習狀況，並利用 mAP（平均精確度）來評估模型在驗證集上的表現，確保辨識準確率達到應用標準。

8. 模型優化與調整

為優化模型，本研究先著手「增加數據量」，由於在測試過程中，發現某些類別的垃圾辨識效果較差，例如部分過度壓縮金屬罐容易被誤判，因此本研究額外收集並標記更多該類別的數據，以提升模型的學習能力。

其次是「增量訓練（Fine-tuning）」，系統在經過一段時間運行後，本研究持續收集新數據並重新訓練模型，以提升系統的學習能力。例如，當新型回收物（如新材質塑膠瓶）出現時，本研究可以手動標記新數據，或使用原始訓練完的模型進行自動標記，並讓 YOLO 進行增量訓練，促使模型不斷修正，達到最佳化程度。

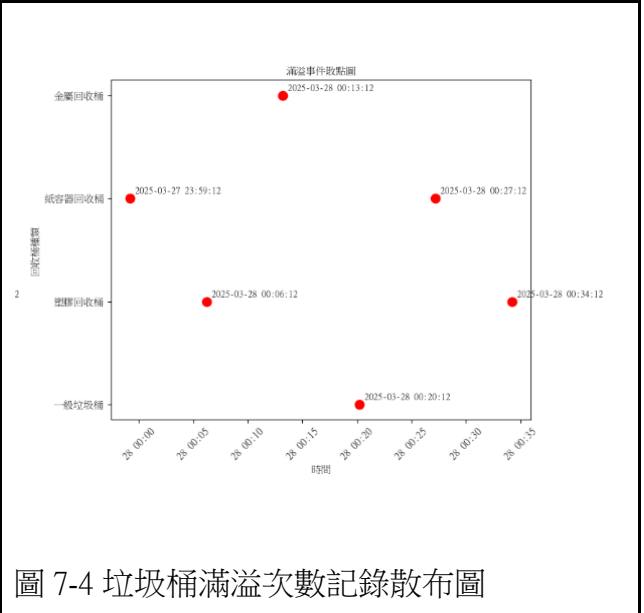
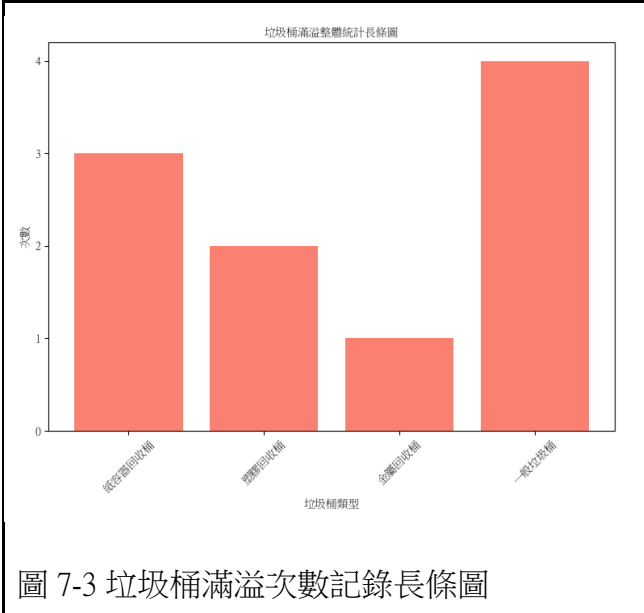
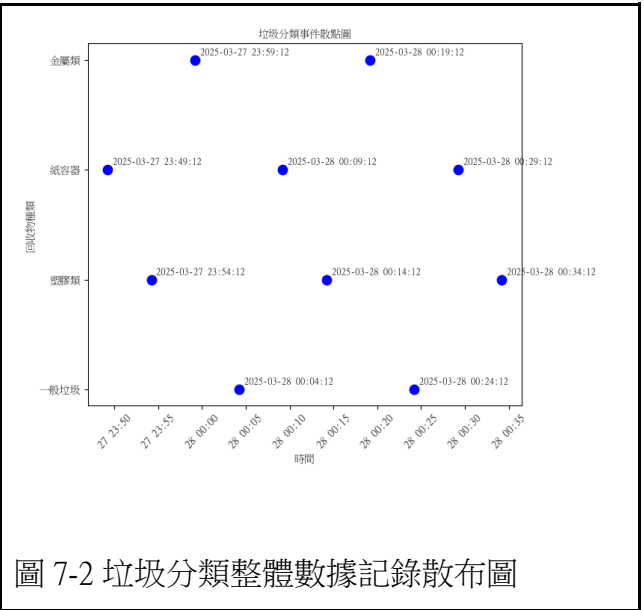
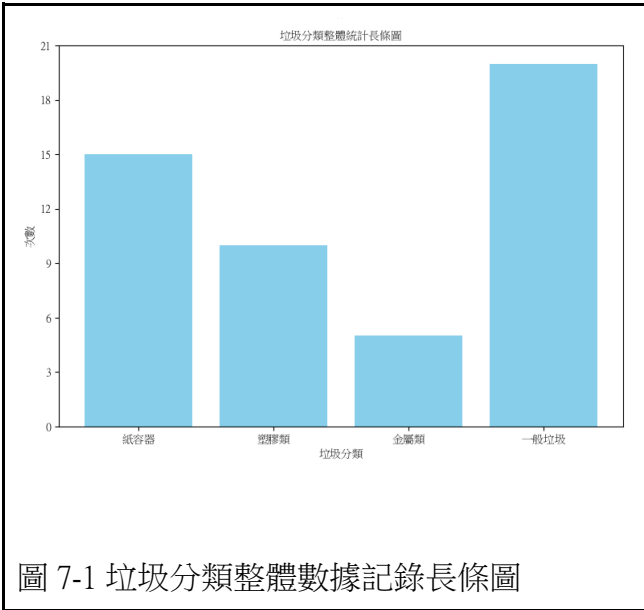
二、系統演算法

當模型訓練出結果後，如何運用視覺辨識模型也是一項重大的挑戰，且如何在實際產品上遇到問題怎麼去做修正也是一道難題。

系統演算法的目標是建立一套智慧垃圾分類與管理系統，在嵌入式裝置 Raspberry 5 實行視覺辨識，以實現垃圾自動分類、回收物搬運、物聯網狀態監控回報與產品的自我增強。

當使用者投入垃圾時，系統會透過攝影機捕捉影像結合燈光效果提示，利用已訓練好的 YOLO 辨識模型判斷垃圾種類，並根據判斷結果控制步進馬達將垃圾移至對應的回收區。同時系統也持續監測各垃圾桶內部的狀態，當檢測到垃圾桶容量達到設定的滿溢門檻時，即會發出警告，並利用 Line notify 即時通知。此外，系統會通過 SQLite 記錄每次垃圾分類與滿溢的數據，作為後續分析與模型增量訓練的依據，以進一步提升辨識準確率，在程式關閉後為了要統計分類結果，透過數據紀錄繪製圖表和進行 Line notify 傳送運行結果統計通知。

表 9 程式繪製圖表釋例



資料來源：研究者自行製作

本研究資源回收系統算法的完整流程圖，如圖 8、圖 9 所示。

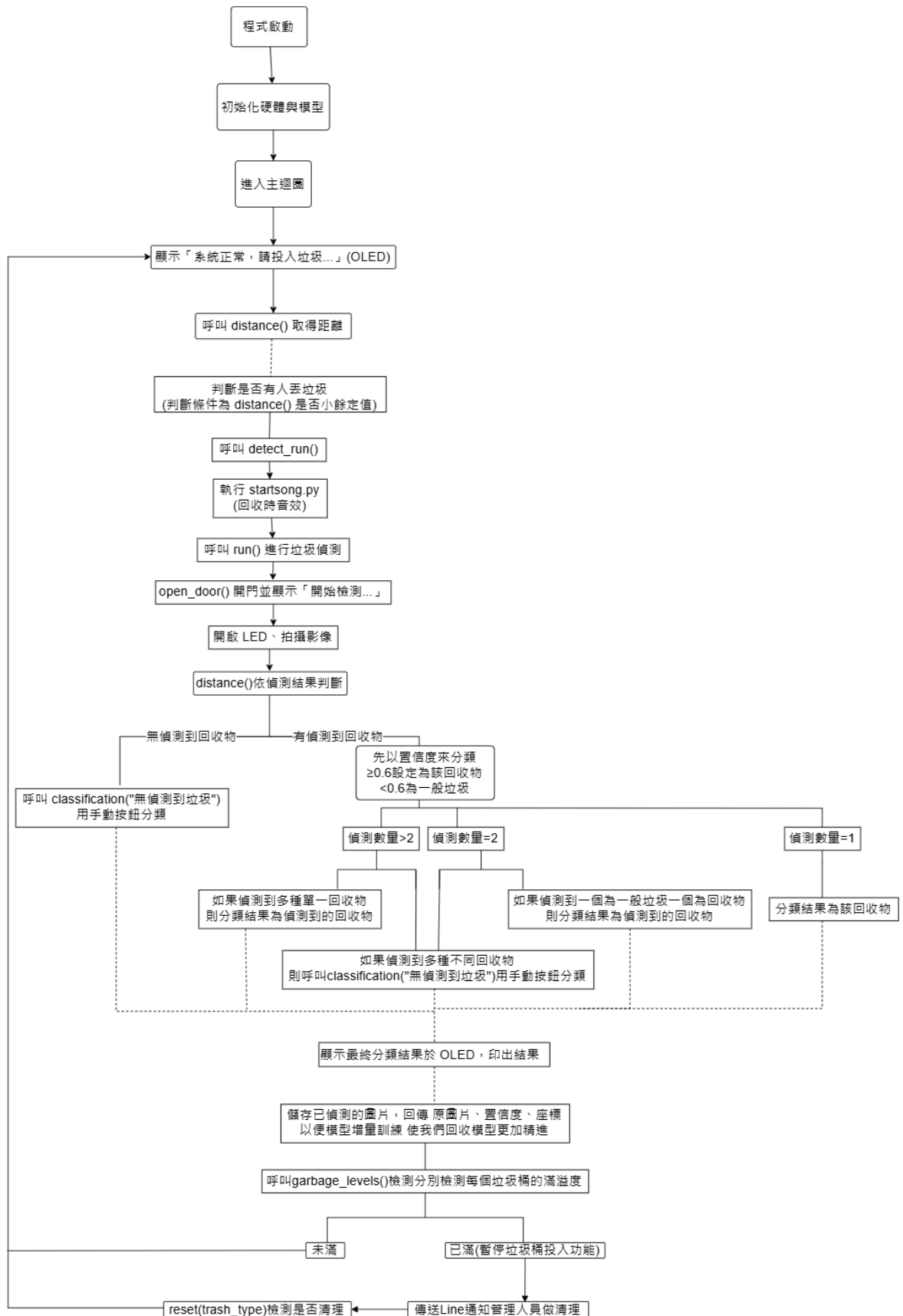


圖 8 資源回收系統之演算法一（資料來源：研究者自行製作）

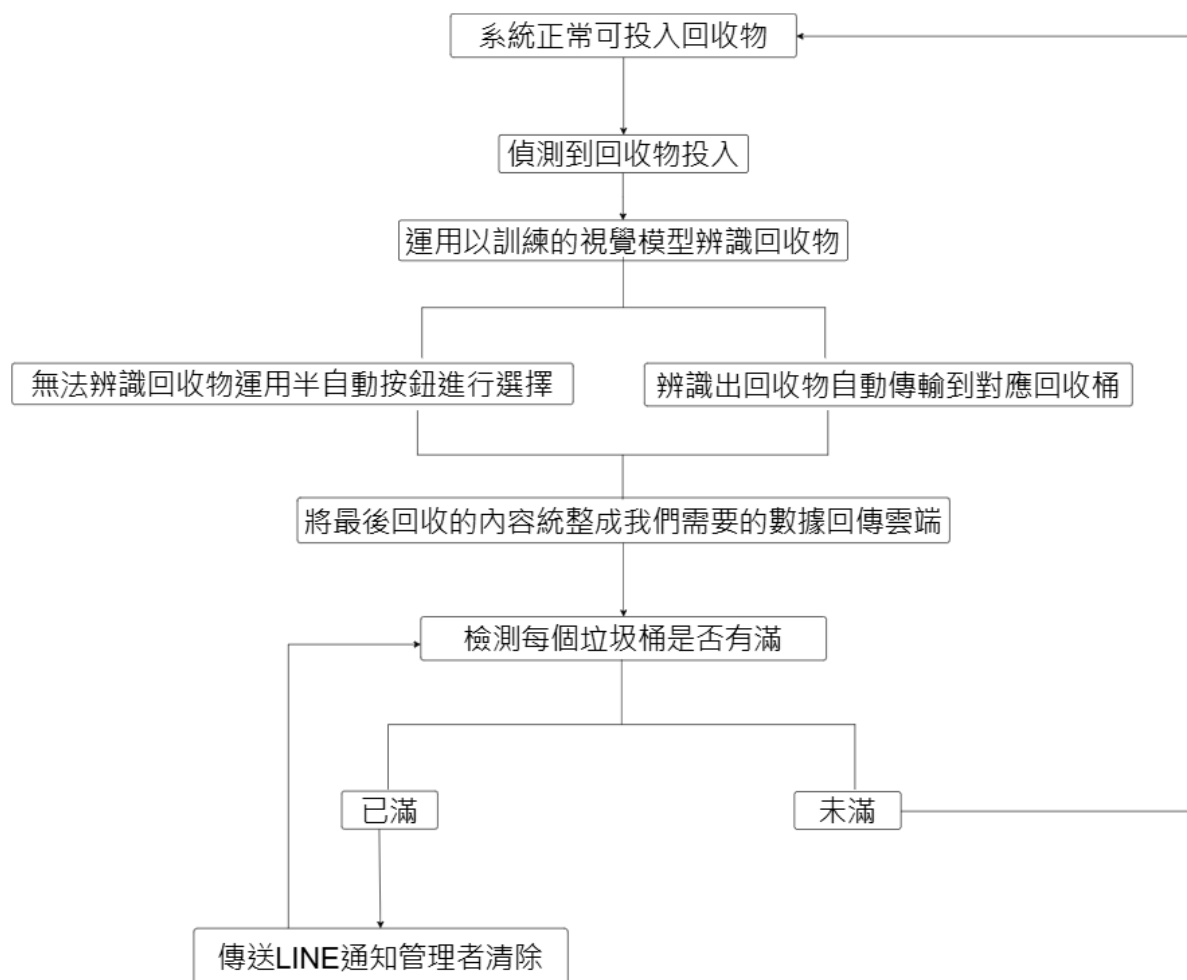


圖 9 資源回收系統之演算法二（資料來源：研究者自行製作）

為達到研究的目標，讓產品可成功完成資源回收任務，演算法開發過程中先將主程式細分為許多小模組，以利逐一設計解決問題。

表 10 演算法函數設計之說明

程式定義函數名稱	功能說明
move(trash)	根據傳入的垃圾類型（「塑膠類」、「紙容器」、「金屬類」或「一般垃圾」），利用步進馬達進行垃圾搬運。依不同類型設定不同的步數（例如：塑膠類採用 4350 步、紙容器 6800 步、金屬類 8900 步、一般垃圾 2000 步），中途呼叫 push() 以觸發推動機構，並改變方向返回初始位置。
push()	控制繼電器動作（透過 GPIO 25），運用電磁鎖模擬推送機構的操作，協助垃圾搬運。
text_display(message)	利用 OLED 顯示器，在螢幕上繪製並顯示傳入的文字訊息，讓使用者能即時看到系統狀態或提示。

waring_choose()	在需要使用者半自動分類時，透過喇叭播放兩次警告音，提醒使用者進行垃圾分類操作。
waring_full()	當垃圾桶滿溢時，透過喇叭播放三次警告音，強調垃圾桶狀態異常，提醒管理人員注意。
connect_db()	建立並返回與 SQLite 資料庫 ("garbage_data.db") 的連線，供後續資料讀寫使用。
open_door()	控制伺服馬達開啟垃圾投入門，同時在 OLED 顯示「開始檢測…」訊息，提示使用者垃圾即將進行辨識。
save_data(category)	根據傳入的垃圾分類 (category)，更新資料庫中 trash_log 表對應分類的次數 (times) 加 1。
save_trash_full(bin_type)	當垃圾桶滿溢時，更新資料庫中 trash_full 表對應垃圾桶類型的滿溢次數 (times) 加 1。
distance()	使用超音波感測器測量距離，並返回以公分計算的距離值，判斷是否有人要丟垃圾。
classification(port)	當自動偵測結果不明確或需要人工確認時，啟動手動分類流程：透過按鈕等待使用者在 10 秒內選擇垃圾類型，若超時則預設返回「一般垃圾」。
run()	系統演算法核心影像偵測函數：調用 open_door() 開啟垃圾門、啟動 LED 與攝影機拍攝影像，再利用 YOLO 模型進行物件偵測。依據置信度 (若大於 0.6) 將結果判定為「金屬類」、「紙容器類」或「塑膠類」，否則歸類為「一般垃圾」。同時回傳拍攝影像、分類列表與詳細偵測資訊 (包含邊界框與置信度)，並將結果影像存檔。
detect_run()	垃圾分類主要流程：先播放開始提示音，再呼叫 run() 取得偵測結果，根據偵測結果數量與一致性決定最終分類。如果偵測結果模糊或矛盾則呼叫 classification() 讓使用者手動確認；若有明確結果則更新資料庫、儲存訓練資料 (呼叫 save_training_data()) 並最後執行滑軌移動 (呼叫 move())。
save_training_data(image, detection, final_classes)	儲存用於增量訓練的資料：若分類為「一般垃圾」則僅存影像；否則將影像存入指定資料夾，並依據最終分類保留相應的邊界框標記資料 (以 YOLO 格式儲存)，供後續模型優化使用。
garbage_levels()	定期檢查各垃圾桶狀態：透過超音波感測器讀取各桶內垃圾高度，若發現某桶距離低於設定的滿溢值 (FULL = 20)，則觸發滿溢警告、發送 LINE 通知並更新資料庫，再呼叫 reset() 持續監控該桶是否被清空。
reset(trash_type)	持續監控指定垃圾桶是否已清空：利用對應的超音波感測器反覆讀取距離，直到讀數大於安全高度 (HEIGHT = 20)，並在清空後於 OLED 顯示提示、發送通知。
send_line(message)	使用 LINE Notify API 發送訊息：透過 HTTP POST 將傳入訊息發送到指定 LINE 帳號，通知管理人員系統狀態、警告或運行報告。
get_run_data()	從 SQLite 資料庫中讀取垃圾分類 (trash_log) 與垃圾桶滿溢 (trash_full) 的運行數據，並回傳查詢結果，供報告生成使用。

generate_report()	根據 get_run_data() 取得的數據，生成一份統計報告，內容包括垃圾分類次數與垃圾桶滿溢次數，報告將作為運行結果通知發送給管理人員。
clear_old_data()	重置資料庫中的運行數據：將 trash_log 與 trash_full 表中的次數欄位全部設為 0，為下一輪運行做準備。
trigger_training()	在系統終止時觸發 YOLO 模型的增量訓練：執行外部指令進行訓練，並在訓練成功後呼叫 update_model() 更新最新模型。
update_model()	檢查指定資料夾中 YOLO 訓練圖片數量，若超過 10 張則將這些圖片上傳至 Google Drive（使用已驗證的 Drive API 與指定的資料夾 ID），上傳完成後刪除本地檔案，以釋放空間並統整訓練數據。
init_db()	初始化 SQLite 資料庫：若 trash_log 與 trash_full 表不存在則建立，並插入預設的垃圾分類（如「紙容器」、「塑膠類」、「金屬類」、「一般垃圾」）及垃圾桶種類（例如「紙容器回收桶」、「塑膠回收桶」等），通常在系統部署時執行一次。
create_drive_folder(folder_name, parent_folder_id)	在 Google Drive 指定資料夾下查找或建立同名資料夾，返回該資料夾 ID。
upload_folder(local_folder, drive_parent_folder_id)	將本地指定資料夾中的所有檔案上傳至 Google Drive 中對應的資料夾。
upload_to_drive()	檢查本地標註檔案數量，若超過條件則上傳影像與標註資料到 Google Drive，並清空本地資料夾。
generate_charts()	從資料庫讀取事件資料，利用 matplotlib 產生時間點散佈圖及累積次數圖，並儲存圖表於指定資料夾。

資料來源：研究者自行設計

因為版面限制，僅擷取較重要的系統演算法作詳細說明。

在回收物品判斷流程中，本研究設計專屬的二階段式演算法，以提升整體分類準確率。整體流程可分為兩個步驟：第一步，由相機擷取回收物影像後，導入 YOLO 模型進行初步辨識，並根據 YOLO 模型所輸出的置信度，進行初層分類篩選。第二步，針對第一階段辨識結果，進行進一步分析與修正，並增加按鈕分類來輔助判斷不出來的物件，以降低回收物判斷的失誤率。此二階段策略可有效補足傳統單一分類模型對於模糊邊界樣本的誤判問題，並顯著提升回收分類模型的整體精準度與實用性。

表 11 擷取影像並導入模型的演算法介紹

回收物拍攝演算法詳細介紹	
def run(): open_door()	在程式中因為要配合許多硬體設備例如：垃圾桶翻蓋開關、步進馬達的位移、相機的

<pre> led = LED(18) led.on() time.sleep(1) cap = cv2.VideoCapture(0) hold = 0.6 ret, frame = cap.read() led.off() results = model(frame) detect = [] detection = [] class_mapping = {0: '金屬類', 1: '紙容器', 2: '塑膠類'} for result in results[0].boxes: class_id = int(result.cls) confidence = float(result.conf) class_name = class_mapping.get(class_id) final = class_name if confidence >= hold else '一般垃圾' detect.append(final) bbox = result.xyxy[0].tolist() detection.append({ "class_name": class_name, "confidence": confidence, "bbox": bbox }) nowtime = datetime.now().strftime("%Y%m%d_%H%M%S") save_img = f'/home/pi/Desktop/yolo_project/run_img/{nowtime}.jpg' cv2.imwrite(save_img, results[0].plot()) cap.release() cv2.destroyAllWindows() os.system("python3 music.py &") return frame, detect, detection </pre>	<p>影像擷取、音樂的撥放等，也為了符合SDG11（永續城市與社區）LED 只開啟在拍攝的時間點，使產品不持續耗電。</p> <p>攝影機讀取影像： 使用 OpenCV 的 cv2.VideoCapture(0) 來開啟預設攝影機。呼叫 cap.read() 讀取一幀影像，並將影像資料存入 frame 變數。</p> <p>影像辨識與物件檢測： 呼叫 model(frame) 將拍攝到的影像導入以訓練好的 YOLO 模型進行辨識，取得 results。定義一個 class_mapping 字典，將數字類別編號對應到實際的垃圾分類（0 對應「金屬類」、1 對應「紙容器」、2 對應「塑膠類」）。設定一個置信度門檻 hold = 0.6，用以判斷偵測結果準確度。逐一處理每張偵測結果 results[0].boxes 中每個檢測到的物件：取出該物件的類別 ID 與置信度（confidence）。根據 class_mapping 取得對應的類別名稱。若置信度大於或等於門檻值，則將最終結果設為該類別名稱；否則視為「一般垃圾」。將最終結果加入 detect 清單，並同時整理出包含類別名稱、置信度及邊界框（bbox）的詳細資料，存入 detection 清單。</p> <p>影像儲存與後續處理： 以目前的日期與時間生成檔名（格式為 "YYYYMMDD_HHMMSS"），組成圖片路徑。使用 cv2.imwrite() 儲存經模型處理後繪製有偵測框的影像（results[0].plot()）。釋放攝影機資源並關閉所有 OpenCV 視窗。</p> <p>回傳結果： 函式回傳拍攝的影像 frame、偵測到的簡略分類結果 detect（例如「金屬類」、「一般垃圾」等）以及詳細的偵測原始資訊 detection。</p>
--	--

資料來源：研究者自行設計

表 12 精準判斷回收物流演算法介紹

回收物準確判斷演算法詳細介紹	
<pre> def detect_run(): os.system("python3 startsong.py &") print(" 偵測物品掉落，開始檢測...\n") frame, detected, det_info = run() only = True l = len(detected) if detected is None: detected = [] if l == 0: final = classification('無偵測到垃圾') print(" 無偵測結果，跳過儲存訓練資料") else: generally_count = detected.count('一般垃圾') if l == 2 and generally_count == 1: detected.remove('一般垃圾') final = detected[0] elif l > 2: if '一般垃圾' in detected: detected = [d for d in detected if d != '一般垃圾'] l = len(detected) if l > 1: for i in range(1, l): if detected[i-1] != detected[i]: only = False break if only: final = detected[0] else: print(' 檢測失敗\n 請按壓按鈕分類') final = classification('檢測失敗') elif '一般垃圾' in detected: print(' 檢測到一般垃圾如果偵測錯誤請按壓按鈕分類') final = classification(' 檢測到一般垃圾') else: final = detected[0] print(f" 最終分類結果：{final}") log_trash_event(final) </pre>	<p>開始偵測與啟動音效： 執行 <code>os.system("python3 startsong.py &")</code>，透過背景執行播放啟動的提示音。 再螢幕輸出提示訊息「偵測物品掉落，開始檢測...」，告知使用者偵測程序已啟動。</p> <p>物件偵測資料存取： 呼叫前述的 <code>run()</code> 函式，取得拍攝影像（<code>frame</code>）、簡略判斷結果列表（<code>detected</code>）以及詳細偵測資訊（<code>det_info</code>）。 變數 <code>only</code> 初始設為 <code>True</code>，用來判斷多筆偵測結果是否一致。</p> <p>檢查偵測結果數量與內容： 先取得 <code>detected</code> 清單的長度(<code>l</code>)。 若 <code>detected</code> 為 <code>None</code> 則轉為空列表。</p> <p>無偵測結果時： 若數量(<code>l</code>) == 0，則呼叫 <code>classification('無偵測到垃圾')</code>（可能為手動分類或預設處理），並輸出「無偵測結果，跳過儲存訓練資料」的訊息。</p> <p>處理偵測結果邏輯： 若偵測結果存在，先統計 <code>detected</code> 中出現「一般垃圾」的次數。</p> <p>情境判斷： 兩筆結果且其中一筆為「一般垃圾」： 移除「一般垃圾」，將剩下的分類結果作為最終分類。</p> <p>超過兩筆結果： 若清單中含有「一般垃圾」，先移除該類型，再重新計算偵測結果筆數。</p> <p>若移除後的偵測結果有多筆，檢查這些結果是否一致： 若所有結果皆相同（<code>only</code> 為 <code>True</code>），則採用該分類。</p> <p>若結果不一致，則輸出「檢測失敗」訊息，並呼叫 <code>classification('檢測失敗')</code> 讓使用者透</p>

<pre> save_training_data(frame,det_info, final) save_data(final) os.system("python3 endsong.py &") text_display(f" 分 類 結 果 : \n {final}") move(final) </pre>	<p>過按壓按鈕進行分類。</p> <p>若偵測結果中含有「一般垃圾」： 輸出提示訊息，並呼叫 classification('檢測到一般垃圾') 處理分類。 呼叫 log_trash_event(final) 記錄垃圾分類事件，可能存入日誌資料庫或檔案中。 使用 save_training_data(frame, det_info, final) 儲存此次偵測的影像與偵測資料，提供未來重新訓練或優化模型的依據。 呼叫 save_data(final) 儲存最終分類結果。 執行 os.system("python3 endsong.py &") 播放結束音效。 使用 text_display(f"分類結果：\n {final}") 將結果透過文字於螢幕顯示方式呈現。</p> <p>最後呼叫 move(final) 根據最終分類結果控制硬體裝置執行移動（例如分流垃圾到對應的回收桶）。</p>
--	--

資料來源：研究者自行設計

為提升系統運作期間的數據精準度，本研究融合自動化標註演算法：當每次投入回收物後，系統即時擷取原始感測資料，先經由演算法處理後判別出更準確的回收物類別。隨後，系統會自動提取並儲存訓練模型所需的代號、座標及影像資料，累積至設定數量後，自動上傳至 Google Drive 雲端空間。此流程可為日後進行增量式訓練提供資料基礎，持續提升回收分類模型的準確度與系統整體效能。

表 13 自動化數據標註演算法介紹。

自動化標註演算法詳細介紹	
<pre> def save_training_data(image, detection, final_class, manual_classification=False): if isinstance(final_class, list): final_class = final_class[0] if not detection: print("無偵測到任何座標，故不儲存訓練資料") return nowtime = datetime.now().strftime("%Y%m%d_%H%M%S") </pre>	<p>處理輸入資料格式： 函式接受四個參數：影像（image）、偵測結果（detection）、最終分類（final_class）以及一個布林值（manual_classification）用來標記是否進行手動分類。開始時，程式會檢查傳入的 final_class 是否為串列，若是則取出第一個元素，確保接下來處理的分類結果為單一字串。</p> <p>確認有偵測結果： 在進行資料儲存之前，程式會先檢查 detection 是否存在任何偵測到的座標資料。</p>

```

if final_class == "一般垃圾":
    folder_path = "./general/"
    os.makedirs(folder_path, exist_ok=True)
    img_file = os.path.join(folder_path,
f"{nowtime}.jpg")
    cv2.imwrite(img_file, image)
    return
image_folder =

"/home/pi/Desktop/yolo_project/auto/train/images"
label_folder =

"/home/pi/Desktop/yolo_project/auto/train/labels"
os.makedirs(image_folder, exist_ok=True)
os.makedirs(label_folder, exist_ok=True)
img_file = os.path.join(image_folder,
f"{nowtime}.jpg")
cv2.imwrite(img_file, image)
bboxes = [
    d["bbox"] for d in detection
    if isinstance(d["class_name"], str)
    and d["class_name"].strip() ==
final_class.strip()
    and (d["confidence"] >= 0.6 or
manual_classification)
]
if not bboxes:
    return
txt_file = os.path.join(label_folder,
f"{nowtime}.txt")
h, w, _ = image.shape
with open(txt_file, "w") as f:
    for bbox in bboxes:
        x_min, y_min, x_max, y_max = bbox
        x_center = ((x_min + x_max) / 2) / w
        y_center = ((y_min + y_max) / 2) / h
        width = (x_max - x_min) / w
        height = (y_max - y_min) / h
        class_mapping = {'金屬類': 0, '紙容器':
1, '塑膠'          類: 2}
        class_id =
class_mapping.get(final_class.strip(),
-1)
        if class_id == -1:
            continue
        f.write(f'{class_id} {x_center:.6f}

```

如果偵測結果為空，代表沒有任何物件被偵測到，直接中止函式，避免無意義的資料儲存動作。

產生唯一的時間戳記：

為了確保每一次儲存的檔案名稱不會重複，程式會根據目前的日期與時間生成一個時間，後續在命名影像與標註檔時皆會使用這個時間字串。

針對「一般垃圾」的特殊處理：

因為在訓練模型時不會用到一般垃圾的數據。此時會建立一個名為"./general/"的資料夾，並將影像以時間戳記作為檔名存入該資料夾中。

建立儲存訓練資料的資料夾：

當分類結果不是「一般垃圾」時，程式會建立存放訓練影像與標註檔的資料夾（例如影像存放在

"/home/pi/Desktop/yolo_project/auto/train/images"，標註檔放在

"/home/pi/Desktop/yolo_project/auto/train/labels"），確保這些資料夾存在，以便後續存檔。

儲存影像檔：

影像會以時間戳記作為檔名，存入剛剛建立的影像資料夾中，這樣可以保證每次拍攝的影像都能夠被正確記錄下來。

篩選符合條件的偵測框：

接下來，程式會從傳入的 detection 資料中，根據以下條件篩選出符合要求的 bounding boxes（偵測框）：

該偵測結果的 class_name 為字串，且在去除左右空白後與 final_class 一致。

該偵測結果的置信度必須大於或等於 0.6，除非使用了 manual_classification，此時即使置信度不滿足也會被保留。

這個步驟的目的是保證只有與最終分類一致且可靠的偵測框被納入標註資料中。

生成標註檔案（YOLO 格式）：

若有符合條件的偵測框，程式會在標註資料夾中以時間戳記命名生成一個文字檔。接著，根據影像的高度與寬度，將每個偵測框的座標（左上與右下座標）轉換為 YOLO 格

<pre>{y_center:.6f} {width:.6f} {height:.6f}\n")</pre>	<p>式所需的相對位置數值，即計算出中心點座標、寬度與高度（均以影像尺寸做正規化）。</p> <p>此外，程式中定義了一個 class_mapping 字典，用來將文字型的分類（例如「金屬類」、「紙容器」、「塑膠類」）轉換成數字型的標籤。若最終分類在字典中找不到對應的數字標籤，則會印出錯誤訊息並略過該偵測框。符合條件的偵測框資料將依照格式（類別編號與各數值）一行一行寫入標註檔中。</p>
--	---

資料來源：研究者自行設計

在程式中亦運用 SQLite 統計處理結果，並利用 Line notify 做即時回報，這主要是用來記錄兩類數據：垃圾分類記錄表 (trash_log)及垃圾桶滿溢記錄表 (trash_full)，如表 14、表 15 所示。

表 14 資料庫格式一

垃圾分類記錄表 (trash_log)		
<pre>cursor.execute(""" CREATE TABLE IF NOT EXISTS trash_log (id INTEGER PRIMARY KEY AUTOINCREMENT, category TEXT UNIQUE, times INTEGER DEFAULT 0)""")</pre>	id	自動遞增的唯一識別碼
	category	垃圾的種類（紙容器、塑膠類、金屬類、一般垃圾），設為 UNIQUE
	times	該類垃圾的分類次數，初始值為 0

資料來源：研究者自行設計

表 15 資料庫格式二

垃圾桶滿溢記錄表 (trash_full)		
<pre>cursor.execute(""" CREATE TABLE IF NOT EXISTS trash_full (id INTEGER PRIMARY KEY AUTOINCREMENT, bin_type TEXT UNIQUE, times INTEGER DEFAULT 0)""")</pre>	id	自動遞增的唯一識別碼
	bin_type	垃圾桶的種類（紙容器回收桶、塑膠回收桶、金屬回收桶、一般垃圾桶），設為 UNIQUE
	times	該垃圾桶滿溢次數，初始值為 0

資料來源：研究者自行設計

研究中採用超音波解決垃圾分類的時間差問題，並節省 Raspberry Pi 5 GPIO 腳位使用，如下表所示。

表 16，超音波演算法示例

超音波 TRIG 腳位共用	
<pre>TRIG = 5 ECHO_GPIO = [6, 13, 16, 26] for i, echo_pin in enumerate(ECHO_GPIO): with DistanceSensor(echo=echo_pin, trigger=TRIG) as sensor: time.sleep(0.1) distance_val = int(sensor.distance * 100)</pre>	這段程式碼是分別檢測不同垃圾桶滿溢度，為了節省 GPIO 腳位，本研究除了有分別隔開不同的超音波，也將不同的超音波發送時間隔開來，間隔時間為 0.1 秒，因此 ECHO 所收到的回波不會互相影響，可以達到節省 GPIO 腳位的目的

資料來源：研究者自行製作

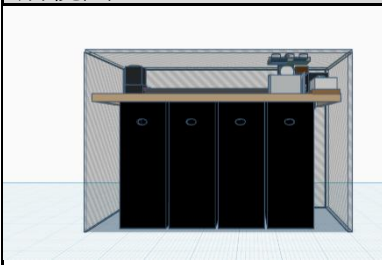
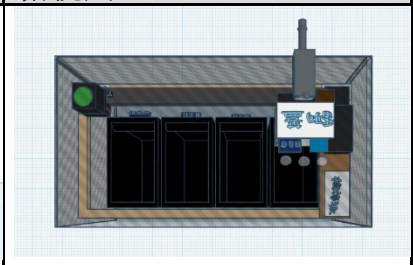
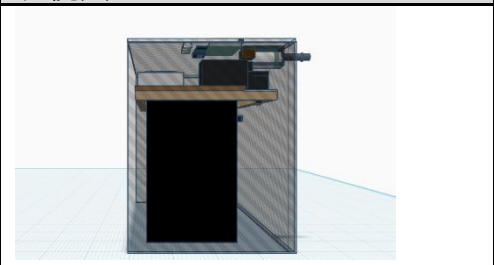
三、系統整合測試

本研運用深度學習，在筆記型電腦使用 GPU(Nvidia GeForce RTX4060)執行 2024 推出的最新 YOLO11 模型進行訓練，並以運算能力已升級的 Raspberry Pi 5 裝置為核心平台，程式撰寫的部份則是使用 Python。由市面上並無現成資源回收裝置或元件可套用，為讓資源回收系統成功運行，因此系統須由研究者根據研究需求，自行設計、組裝與測試。系統設計可分為三個主要階段：3D 設計產品、功能設計及測試修正。

(一) 3D 設計產品

研究者在實作前，為了妥善規劃各種器具的位置，於是先行使用了 Tinkercad 來製作回收桶的 3D 圖，這樣不僅能夠讓產品在實作時更有條理，也能更加了解垃圾桶、外殼及相關設備的預設尺寸，來提升實作時的效率。

表 17 產品 3D 設計圖

前視圖	俯視圖	右視圖
		
圖 10-1 前視圖	圖 10-2 俯視圖	圖 10-3 右視圖

資料來源：研究者自行整理

(二) 功能設計

由於 Raspberry Pi 5 上面的某些特殊腳位具有特定功能，且系統需要外接許多設備，因此研究小組特別製作了相關的器材連接圖，以便在接線時進行確認，確保連接正確並降低接線

錯誤的風險，如下圖所示。

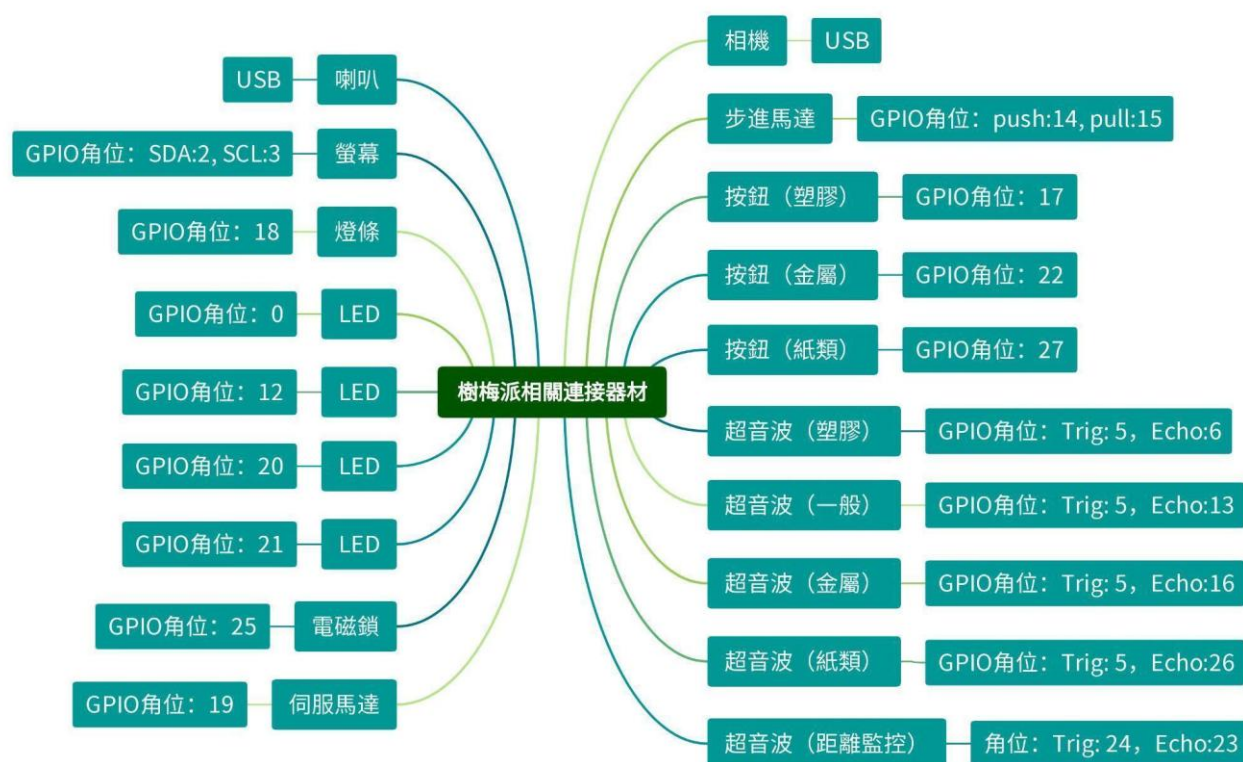


圖 11 資源回收機功能設計（資料來源：研究者自行製作）

（三）測試修正

在運行過程中本研究發現在拍攝照片時，裝置的其它平面以及線路會造成反光，導致取得的照片雜亂，產生未對焦辨識的物件，造成照片辨識準確率下降。經多次測試後，研究發現特別額外框選出一個小區域的範圍，並將周圍噴上黑漆，以避免其他物品對真正辨識物的干擾，有效提高視覺辨識成功機率。

此外，在測試過程中，研究發現沒辦法將滑軌等東西直接固定在 PP 版上，因此改用木材製作框架，以提升整體支撐力。同時，也一併改動了滑軌的高度。然而，由於垃圾桶尺寸已經固定且已完成製作，若不額外進行其他調整，回收物在分類過程中可能會掉落到錯誤的桶子裡面。為了解決此問題，研究者在底部加裝墊高的板子，降低回收物意外掉落至錯誤桶內的風險。

最後，研究原本預設的回收桶大小是長 100 公分、寬 60 公分、高 80 公分，但在查閱科展簡章後，發現科展中心對作品尺寸有所限制，因此本研究將整體大小調整為長 70 公分，寬 60 公分，高 50 公分，然而，在實際完成硬體架設後，發現並不需要那麼多的寬度。因此，在適當縮短底部支撐所有設備的木板長度後，將整體長度改為可適合放下所有硬體設備。

（四）完成實體產品

本研究透過上述設計構想，將整體系統修正為長 70cm、寬 39cm、高 50cm，並順利成功地製造出智慧回收桶的實體裝置。以下是此回收桶的外觀。在回收桶的最前端，本研究選擇用壓克力板，這樣能更清晰的看出在最終分類過程結束後，回收物會掉到的垃圾桶內。在組裝過程中，研究小組針對各個元件的位置以及線路修改調整，藉此確保滑軌運行順暢，也稍微將電線進行整理，以減少線路雜亂對其他零件的影響。此外，本研究也實施了一些外殼上的修飾，使回收桶的整體外觀更加完整，以提升整體質感。完工的實體展示如下圖所示：



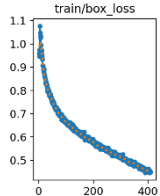
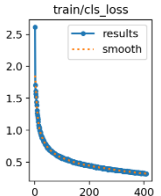
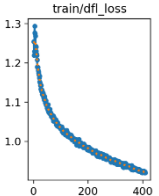
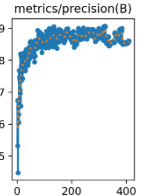
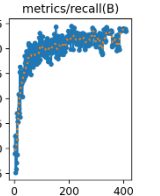
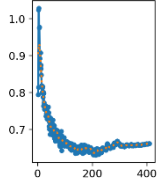
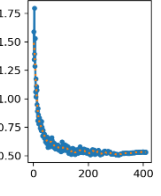
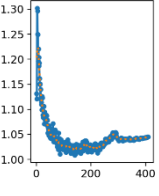
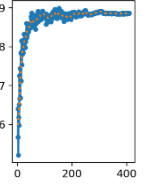
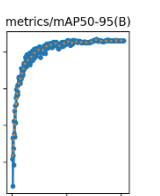
圖 12 資源回收機實體（資源來源：研究者自行製作）

肆、研究結果

一、模型訓練結果

在深度學習 YOLO11s 模型訓練中，本研究輸入訓練集 3720 圖片，驗證集 330 圖片，並使用不同的超參數訓練(Batch size = 16, Learning rate = 0.001, Epochs = 500)加入訓練，數據分析的結果如下表所示。

表 18 YOLO 訓練集與驗證集的分析結果

訓練集分析				
圖 13-1 訓練檢測框損失率 	圖 13-2 分類損失率 	圖 13-3 邊界框精準度損失率 	圖 13-4 檢測準確度 	圖 13-5 目標檢測覆蓋率 
訓練檢測框損失率，根據圖上數據呈現下降的趨勢表示模型學到了更準確的標註框	分類損失率，損失率逐漸減小，表示分類準確率提高	邊界框精準度損失率，損失率下降表明模型預測的標註框越來越精準	檢測準確度，逐漸穩定在 0.8-0.9 是良好的模型表現	目標檢測覆蓋率，數值上升表示模型檢測正確率提升
驗證集分析				
圖 13-6 檢測框位置損失率 	圖 13-7 分類損失率 	圖 13-8 邊界框精準度損失率 	圖 13-9 檢測準確率 	圖 13-10 目標檢測覆蓋率 
檢測框位置損失率，數據損失率下降表明模型在驗證集上表現改善	分類損失率，損失率逐漸減小，表示分類驗證上準確率提高	邊界框精準度損失率，數據呈現下降並趨於穩定，表明驗證集中標註框的預測能力提高	檢測準確率，數據呈現逐漸上升，表示良好的檢測性能提升	目標檢測覆蓋率，數值上升表示模型的檢測品質及能力逐漸上升

資料來源：研究者自行製作

二、回收物檢測結果置信度分析

YOLO11s 模型訓練結束後，本研究實際部署在資源回收系統機器上測試，並以各十個不同物品（金屬、紙容器、塑膠）進行辨識分類。辨識的結果發現，YOLO11s 對金屬物品的檢測情形整體表現較為穩定，置信度大多維持在 0.8 以上；在紙容器物品的檢測方面，置信度可維持在 0.9 以上；塑膠物品的檢測方面置信度大多維持在 0.6 以上，但置信度的波動較大，特別是透明塑膠物品（因光線穿透影響）的置信度較低。研究結果整理如下表所示：

表 19 不同種類回收物的檢測結果



圖 14-1 金屬物品檢測情形

金屬物品的檢測情形整體表現較為穩定，置信度大多維持在 0.8 以上，但在部分變形較嚴重的金屬罐中，可能有形變導致特徵不明顯的情況產生，或者物體表面反光因而影響視覺特徵，使模型判斷置信度降低。除了形狀外，背景狀況也可能導致誤判，例如大理石桌面可能因為反光或顏色接近金屬，使模型辨識力下降。



圖 14-2 塑膠物品檢測情形

在塑膠物品的檢測中，可觀察到置信度的波動較大，大部分在 0.7 至 0.85 之間，特別是透明塑膠物品的辨識過程，容易因光線反射或背景透光而影響模型的判讀結果。此類原因推測於透明材質的邊界特徵較模糊，且光線穿透後背景影響更大，導致模型難以準確識別物體的輪廓。因此，在進行檢測時，應盡可能減少光源造成的反光，並確保背景顏色統一，以降低環境對模型判讀的影響。

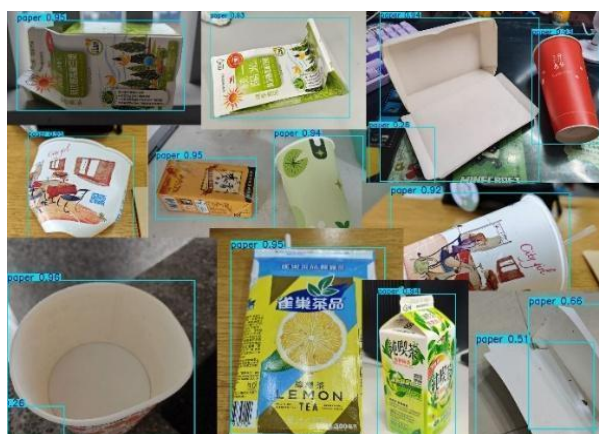


圖 14-3 紙容類檢測情形

在紙容器物品的檢測方面，模型整體表現相當穩定，大多數檢測結果的置信度皆維持在 0.9 以上，即使是開口朝上的紙杯或不同背景環境，也未出現明顯的分類誤差或置信度下降。但在紙盒的檢測中，仍可觀察到置信度略低的情況，這可能受到物品表面的反光、陰影所影響，或背景顏色與紙盒相近，導致邊界特徵不夠明顯，從而影響模型的判斷。

資料來源：研究者自行整理

伍、研究討論

一、演算法的問題

程式執行遇到疑難雜症不少，整理如下表所示：

表 20 系統演算法遭遇問題清單

狀況	問題點	說明與建議
僅標註置信度大於 0.6 的 bbox	save_training_data 中如果沒有 bbox，會寫入圖片但無標註	後續再訓練時這些圖片會浪費空間且對模型無幫助，可考慮加入自動上傳供人工標註。
系統過於依賴影像設備穩定性	開始攝影後只取一張影像，不再持續確認是否偵測成功	若第一張影像曝光不足、模糊、未拍到垃圾，將導致整體失敗
全域變數未統一集中設定	因為程式碼是持續在更新，所以 GPIO 編號、置信度、音樂檔案路徑、分類對應表等散落在程式各處	不利於維護與調整
Google Drive 上傳	上傳後會刪除本地資料，若上傳失敗則資料遺失	改進方式：上傳成功後再刪除，並保留一份壓縮包備份
報表自動傳送	因為在測試所以 LINE Notify 僅在終止時推送一次資料報表	改進方式：可考慮定時（每日/每 10 筆分類）自動推播一次摘要

資料來源：研究者自行設計

二、回收物紀錄問題

面臨數據紀錄時的覆蓋問題，研究採用下表方式解決：

表 21 數據記錄演算法

<pre>default_trash_types = ["紙容器", "塑膠類", "金屬類", "一般垃圾"] for category in default_trash_types: cursor.execute("INSERT OR IGNORE INTO trash_log (category, times) VALUES (?, 0)", (category,)) default_bins = ["紙容器回收桶", "塑膠回收桶", "金屬回收桶", "一般垃圾桶"] for bin_type in default_bins: cursor.execute("INSERT OR IGNORE INTO trash_full (bin_type,times) VALUES (?, 0)", (bin_type,))</pre>	<p>在測試過程中，本研究發現數據覆蓋的問題，導致所有回收物的計數只會累加到同一類別，而無法正確區分各類回收物。為了解決這個問題，本研究在建立資料表時，預先設定了固定的回收物類別，確保每種回收物都有對應的紀錄，最後數據整理只需要找相對應的回收物做統整即可。</p>
---	--

資料來源：研究者自行設計

三、回收桶設計問題

垃圾桶設計滑軌控制方式，電磁鎖及物品移動時，本研究遇到當回收物以特定角度掉落至智慧回收桶內的時候，在辨識完成後，滑軌開始移動時，由於回收物支撐點在壓克力板上，而導致回收物在剛出壓克力板後就會掉落到桶子內，因此回收物無法到正確的地點，本研究目前還沒有想到能完美解決這個問題的方案，但想嘗試更改放置回收物平台以及壓克力這兩者的傾斜度，看問題能否解決。

四、滑軌運行秒數討論

垃圾的物品在 YOLO 即時視覺辨識並正確分類後，利用滑軌設備，投入到分類垃圾桶後，研究撰寫程式計算秒數，並以全自動/半自動模式情況下，統計資源回收分類所需花費的最長時間(平均數)，整理如下表所示。

表 22 資源回收分類處理時間

模式	種類	運行時間
全自動	金屬	19~20 秒
	紙容器	17~18 秒
	塑膠	14~15 秒
	一般垃圾	若偵測到一般垃圾，會自動跳至半自動模式
半自動	金屬	15~26 秒
	紙容器	20~31 秒
	塑膠	15~26 秒
	一般垃圾	22~23 秒

資料來源：研究者自行製作

上述統計秒數的差距，主因是每個垃圾桶離平台之間距離都不同，距離從近到遠分別是金屬、紙容器、塑膠與一般垃圾四種分類，因此時間也是照這個順序遞增，而下方半自動執行的秒數差異較大是因為在使用時，本研究的設置是讓使用者在 10 秒內按按鍵輔助分類，否則會掉入到一般垃圾桶，因此一般垃圾在半自動執行的秒數間隔才會比其他的短，但相對的，它只能等程式跑完才能進行反應，因此秒數略長。

五、回收桶的容量管理

一開始檢測回收桶有無滿溢的方案是將超音波放置在垃圾桶的最後方，但在研究中發現它只能測量到正下方的滿度，因此後面將超音波改架設在垃圾桶約八分滿的位置，當超音波的數值浮動時，則代表有物品超越了那一條滿溢線，就會發出警示，並停止接收其餘回收物

的功能。未來想要嘗試將超音波加裝在平台下方，讓它隨著平台移動、到定位點傾倒回收的同時，也一併測量垃圾筒的滿度，這樣測量的位置跟垃圾下落的主要位置比較相似。

六、光線影響與反光調適

本研究最初架設該機器時，並沒有產生材質辨識錯誤的問題，但後來為了防止垃圾掉到錯誤位置，本研究加裝了壓克力板，卻意外發現這個舉動導致物品在不同光線下的材質辨識結果產生變化，本研究推測是因為壓克力板造成了反光的問題。

因此本研究透過不斷的調整燈條大小、位置，以及光線明暗程度，來達到解決問題的功效，避免類似問題再度發生，本研究還自行收集了多種回收物，並記錄它們在回收桶內由相機拍攝的影像，以擴充模型的訓練資料，確保未來能更準確地處理類似情境。

七、垃圾分類提醒

在給其他人進行測試的時候，本研究收到了一些回饋，像是螢幕太小，有時候會沒有注意到最終的分類結果、進行按鈕分類的提醒與垃圾桶已滿的提示，因此本研究加裝了喇叭和燈光，來進行聲音及視覺上的二次提示，希望能讓使用者了解最終的分類結果與警示。

八、偵測後數據庫的統整和上傳資料問題

在回收桶運行完資料的統整以及數據上傳時，可能碰到沒有網路或是資料上傳錯誤的情境，在這方面研究小組設計分段執行，如果資料無法上傳，系統內的資料就不會自動刪除，等下次再一併上傳即可，可避免資料沒有存入雲端就被刪掉問題。

陸、研究結論與建議

一、研究結論

本研究利用深度學習 YOLO11s 視覺辨識一般垃圾或資源回收類，並藉由優化模型訓練的結果，實際部署在 Raspberry 5 平台上，以智慧化與自動化的方式減少資源回收的錯誤與人力負擔，同時也可收集執行的數據，重新訓練模型，以提高資源回收的辨識率。此外，研究為了解即時運行狀況，使用物聯網，以 Line notify 即時提供數據與訊息以及自動上傳數據到 Google Drive，以供管理單位即時問題、統計數據和增強訓練模型，並且為了結合 SDG10（減少不平等），研究者透過音效提示跟燈光顯示，使不同族群可以使用到本產品，讓使用者更容易辨別回收物。從研究的結果，本研究「利用深度學習實作資源回收系統之設計」已取得初步成功，不管從 mAP 達 0.9 以上，或是實際運行的平均時間為 15~30 秒，效率來看已獲得不錯的成功。

二、研究建議

（一）儘管現在本研究現在使用的深度學習卷積神經網路已經能夠準確辨識出複合型回收物，但因為硬體設備限制本系統目前無法將它進行拆解，因此希望未來能再從這部分下去完善本研究的系統。

（二）本系統旨在推動聯合國永續發展目標（SDGs），但目前仍依賴外部電源供應。為了進一步提升能源效率與環境永續性，本研究希望改採太陽能供電方案，以減少對傳統電力的依賴，並提升系統的自主運行能力。

（三）目前系統可以發展壓縮回收物的功能，但考量到電磁鎖前方的壓克力板可能無法承受其動力，為確保結構穩定性，因此決定暫不加此功能。

（四）Raspberry Pi 5 長時間高負載運行下會產生熱量，若沒有適當的散熱措施，可能會導致系統過熱；在頻繁寫入日誌或資料的情況下，長時間的讀寫操作可能會導致 SD 卡損壞；此外，若要將本系統實際部署室外環境時，則必須考慮更多的因素，例如：網路、氣候、電源與安全因素等，才可因應更多不同的情境。

柒、參考文獻資料

- [1] The Global Goals. (2025). 17 項永續發展目標 . <https://globalgoals.tw/>
- [2] 統奕（2025）。iCIRCLE 高效智慧回收機。
https://www.ppi.com.tw/article_d.php?lang=tw&tb=4&id=243
- [3] Darwin.（2022）。SSD: Single shot multibox detector
<https://darren1231.pixnet.net/blog/post/354771550>.
- [4] Ultralytics. (2025). Ultralytics YOLO11 .
<https://docs.ultralytics.com/zh/models/yolo11/>.