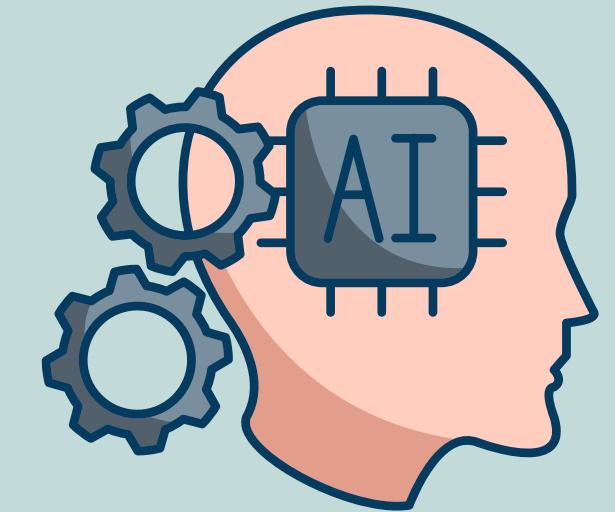
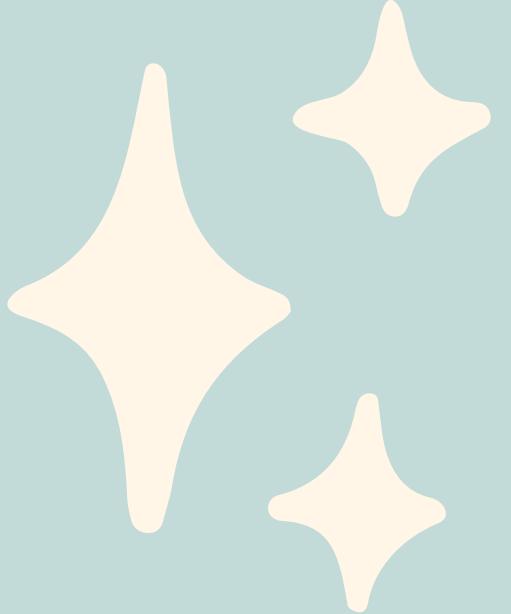


# Lab 4-Efficientdet

第四組-四資管AI三A-B11123224-余雨芹





# TABLE OF CONTENT

01. 模型介紹

02. PC 端 模型訓練 & 推理

03. EDGE 端 操作流程

04. EDGE 端 驗證結果

05. 遇到問題與解決

06. 心得

# 01.模型介紹

## EfficientDet 模型介紹

- EfficientDet 是一種高效能物件偵測模型。

特點：

- 高效能與低運算成本平衡。
- 支援多種規模 (Lite0 ~ Lite3)。
- 適合邊緣設備與資源有限環境。

核心技術

- EfficientNet Backbone：高效的特徵提取，搭配複合式縮放 (Compound Scaling) 提升效能。
- BiFPN (雙向特徵金字塔網路)：提升對不同尺寸物件的偵測能力，減少不必要運算。
- 複合式縮放：同時調整骨幹網路、特徵融合與影像大小，全面優化。

這次Lab4使用的規模 - EfficientDet Lite0

特性：

- 最輕量級 EfficientDet 模型。
- 適用於資源受限設備 (如 Raspberry Pi、行動裝置)。
- 提供即時推理能力，降低計算延遲。

## 數據準備

- 輸入：

- 影像資料夾 (image)。
- 標註檔案 (Pascal VOC 格式 xml)。

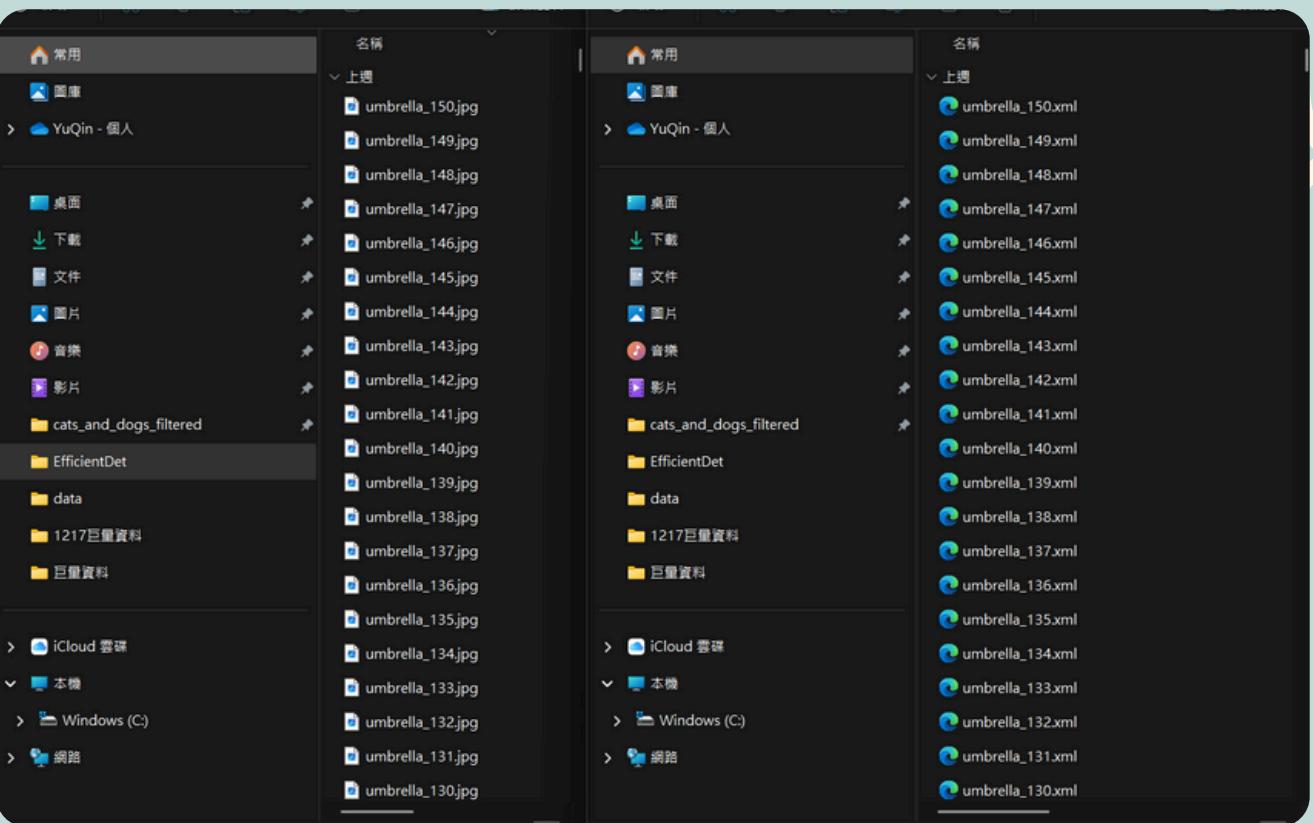
- 資料分割：

- 訓練集、驗證集、測試集按比例分割 (60%/20%/20%)。
- 使用函數 `split_dataset()` 實現。

- 標籤設定：

- 預定義標籤集合 : `["M", "K", "S", "easycard", "umbrella"]`。

## 輸入



## 資料分割

```
1 # 將資料集分割成訓練集、驗證集和測試集
2 train_dir, val_dir, test_dir = split_dataset(
3     images_in,          # 原始影像資料夾的路徑
4     annotations_in,      # 原始標註檔案 (Pascal VOC 格式 XML) 的資料夾路徑
5     val_split=0.2,        # 驗證集的比例 (20%)
6     test_split=0.2,       # 測試集的比例 (20%)
7     out_path=split_out_dir # 分割後資料集輸出的目錄路徑
8 )
```

## 標籤設定

```
1 label_map = ["M", "K", "S", "easycard", "umbrella"]
```

# 02. PC端 模型訓練&推理

## 模型訓練

### 1. 模型規範：

- 使用 `EfficientDetLite0Spec()` 作為模型骨幹架構。

### 2. 訓練配置：

- Epochs: 80, Batch Size: 16。
- 訓練整個模型 (`train_whole_model=True`)。

### 3. 資料載入：

- 使用  
`tflite_model_maker.object_detector.DataLoader`  
載入 Pascal VOC 格式數據。
- 資料快取 (`cache_dir`) 優化性能。

### 4. 訓練函數：

- 使用 `object_detector.create()` 完成模型訓練。

1

```
1 spec = object_detector.EfficientDetLite0Spec()
```

```
1 # 創建並訓練 EfficientDet 模型
2 model = object_detector.create(
3     train_data=train_data,
4     model_spec=spec,
5     validation_data=validation_data, # 驗證數據集，用於訓練過程中的性能評估
6     epochs=80, # 訓練的總迭代次數 (Epoch)
7     batch_size=16, # 每次訓練處理的樣本數量 (Batch Size)
8     train_whole_model=True, # 是否訓練整個模型 (包含骨幹網路和其他層)
9     do_train=True # 執行訓練過程 (如果為 False，則僅創建模型不進行訓練)
10 )
```

```
1 # 加載訓練數據集，格式為 Pascal VOC
2 train_data = object_detector.DataLoader.from_pascal_voc(
3     os.path.join(train_dir, "images"), # 訓練集影像的路徑
4     os.path.join(train_dir, "annotations"), # 訓練集標註檔案的路徑
5     label_map=label_map, # 標籤對應的映射表
6     cache_dir=".cache_data_ta/train", # 快取資料的路徑，用於提高加載效率
7     num_shards=1, # 數據分片數量 (通常設為 1)
8     max_num_images=3000 # 訓練集中最多載入的影像數量
9 )
10
11 # 加載驗證數據集，格式為 Pascal VOC
12 validation_data = object_detector.DataLoader.from_pascal_voc(
13     os.path.join(val_dir, "images"), # 驗證集影像的路徑
14     os.path.join(val_dir, "annotations"), # 驗證集標註檔案的路徑
15     label_map=label_map, # 標籤對應的映射表
16     cache_dir=".cache_data_ta/validation", # 快取資料的路徑，用於提高加載效率
17     num_shards=1, # 數據分片數量 (通常設為 1)
18     max_num_images=1000 # 驗證集中最多載入的影像數量
19 )
```

# 02.PC端 模型訓練&推理

## 模型評估

### 模型評估

- 使用測試集驗證模型性能，輸出準確率與損失指標。



```
1 model.evaluate(test_data)
```

## 模型匯出

### 模型匯出

#### 匯出三種格式：

- TensorFlow Lite 模型 (.tflite)。
- 標籤檔案 (.txt)。
- SavedModel 格式 (.pb)。



```
1 model.export(  
2     export_dir='./export',  
3     tflite_filename='efficientdet.tflite',  
4     label_filename='labels.txt',  
5     saved_model_filename='model',  
6     export_format=[  
7         ExportFormat.TFLITE,  
8         ExportFormat.LABEL,  
9         ExportFormat.SAVED_MODEL  
10    ]  
11 )
```

# 模型匯出的目錄  
# 匯出的 TensorFlow Lite 模型檔案名稱  
# 匯出的標籤檔案名稱  
# 匯出的 SavedModel 格式目錄名稱  
# 指定匯出的格式  
# TensorFlow Lite 格式  
# 標籤檔案格式  
# TensorFlow SavedModel 格式

# 03. Edge端 操作流程

## 1. 優化.tflite-> xxx\_edge\_tpu.tfliteCompile for EdgeTPU.ipynb

1

```
1 # 使用環境變數設定 TFLite 模型的檔案名稱
2 # TFLITE_FILE 是變數名稱，值為 'efficientdet.tflite'，即 TFLite 模型檔案的名稱。
3
4 %env TFLITE_FILE=efficientdet.tflite
```

2

```
1 # $TFLITE_FILE 是環境變數，表示要編譯的 TFLite 模型檔案名稱。
2 # 編譯後的模型檔案通常命名為 <原檔名>_edgetpu.tflite
3 ! edgetpu_compiler $TFLITE_FILE
```

3

```
1 # 安裝 Edge TPU 編譯器
2 ! sudo apt-get install edgetpu-compiler
3
4 # 使用 Edge TPU 編譯器將 TFLite 模型優化為 EdgeTPU 格式
5 ! edgetpu_compiler '/content/drive/MyDrive/Colab/lab4-2/export/efficientdet.tflite'
6
```

4

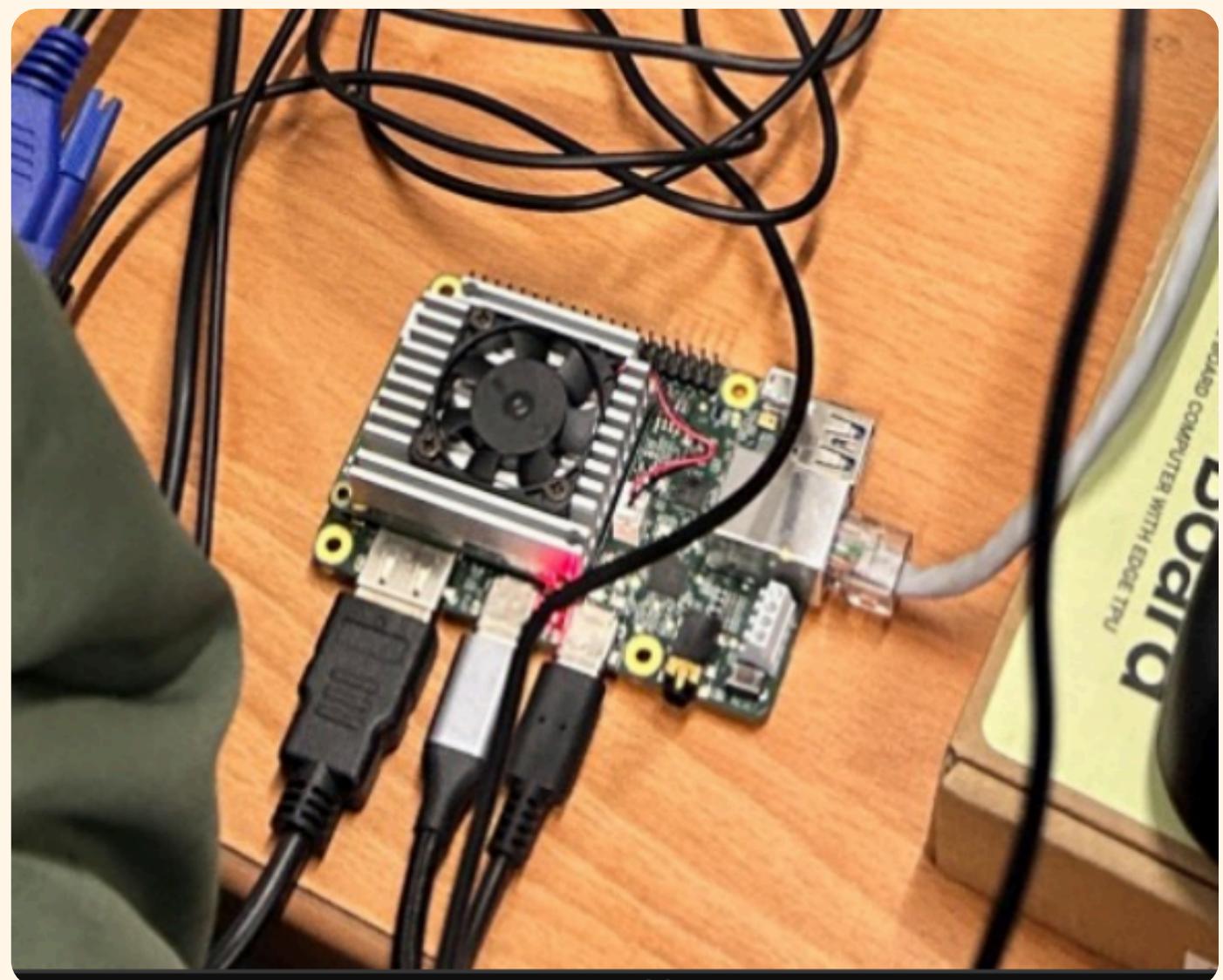
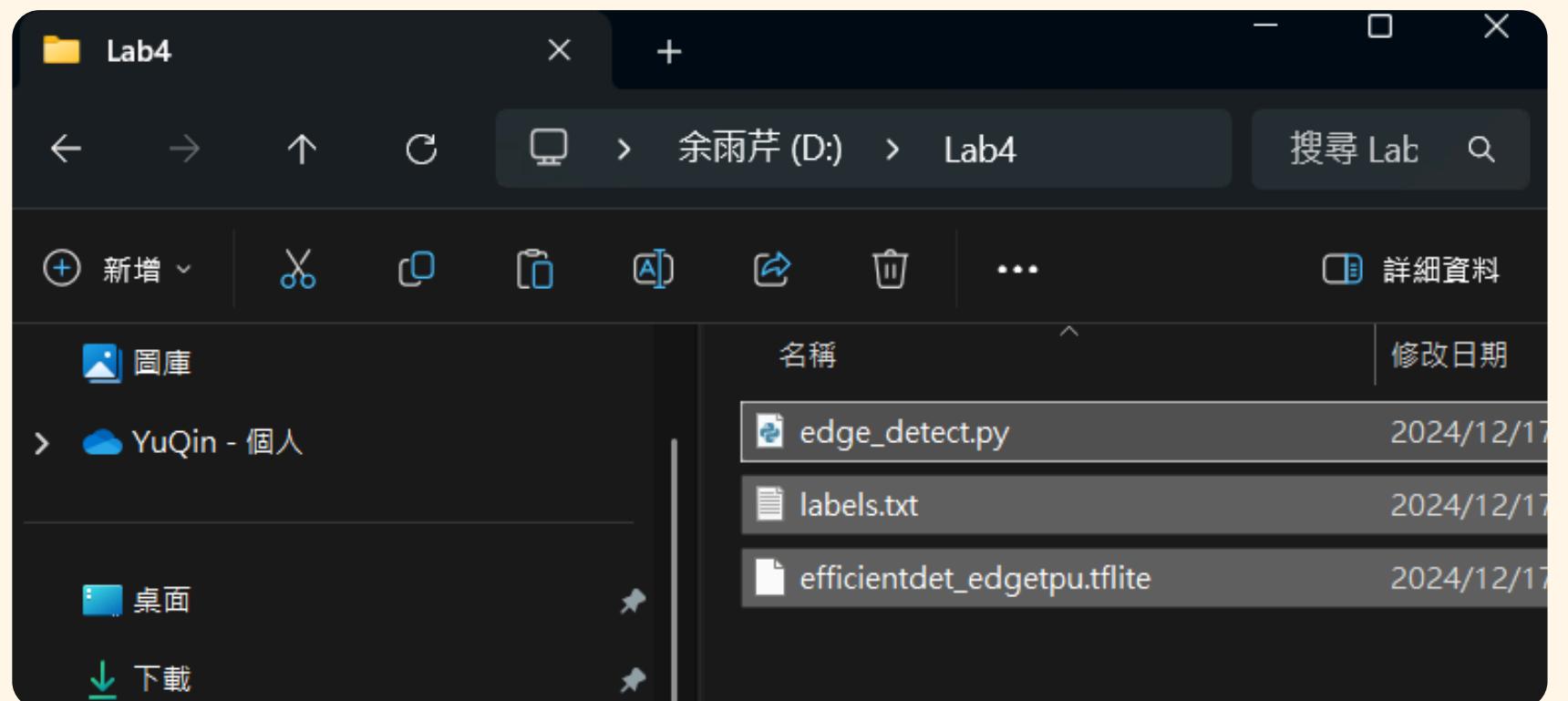
```
1
2 # 獲取環境變數 TFLITE_FILE 的檔名 (不含副檔名)
3 name = os.path.splitext(os.environ['TFLITE_FILE'])[0]
4
5 # 下載 EdgeTPU 優化後的模型檔案
6 files.download(str(name + '_edgetpu.tflit
```



# 03. Edge端 操作流程

## 2. 板端測試驗證模型效果edge\_detector.py

- 將需要的檔案放入USB



- 將Dev board接上螢幕、滑鼠和鍵盤

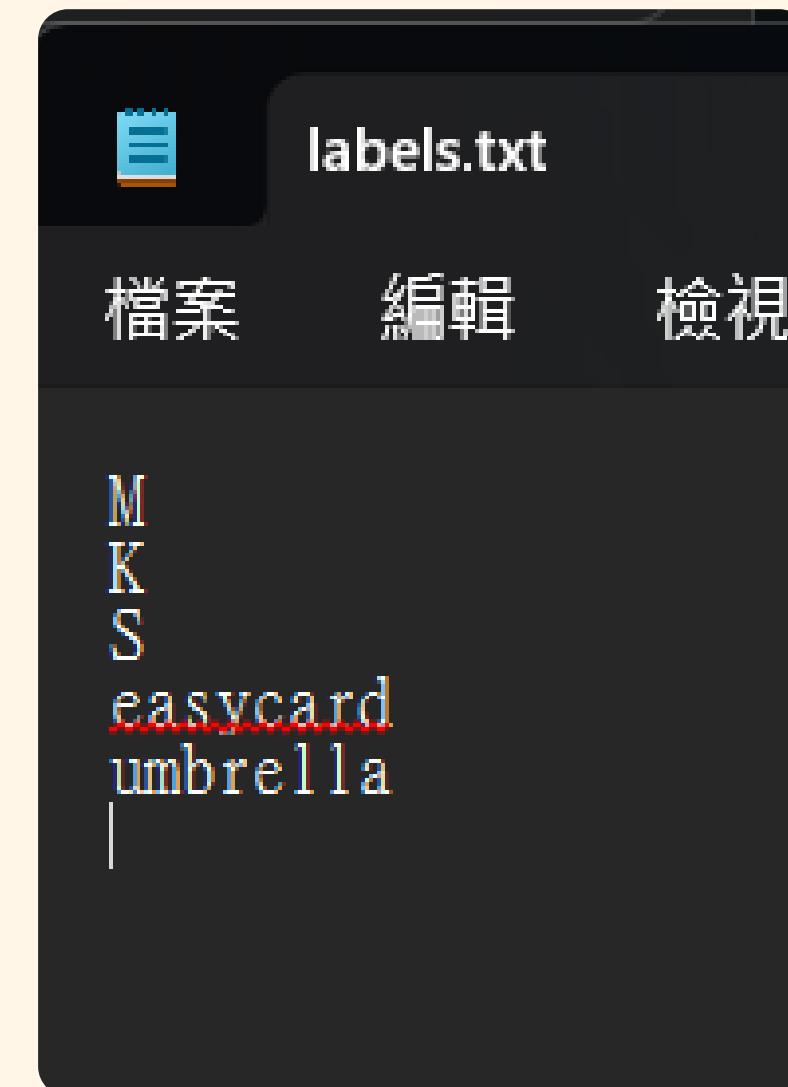
# 03. Edge端 操作流程

## 2. 板端測試驗證模型效果edge\_detector.py

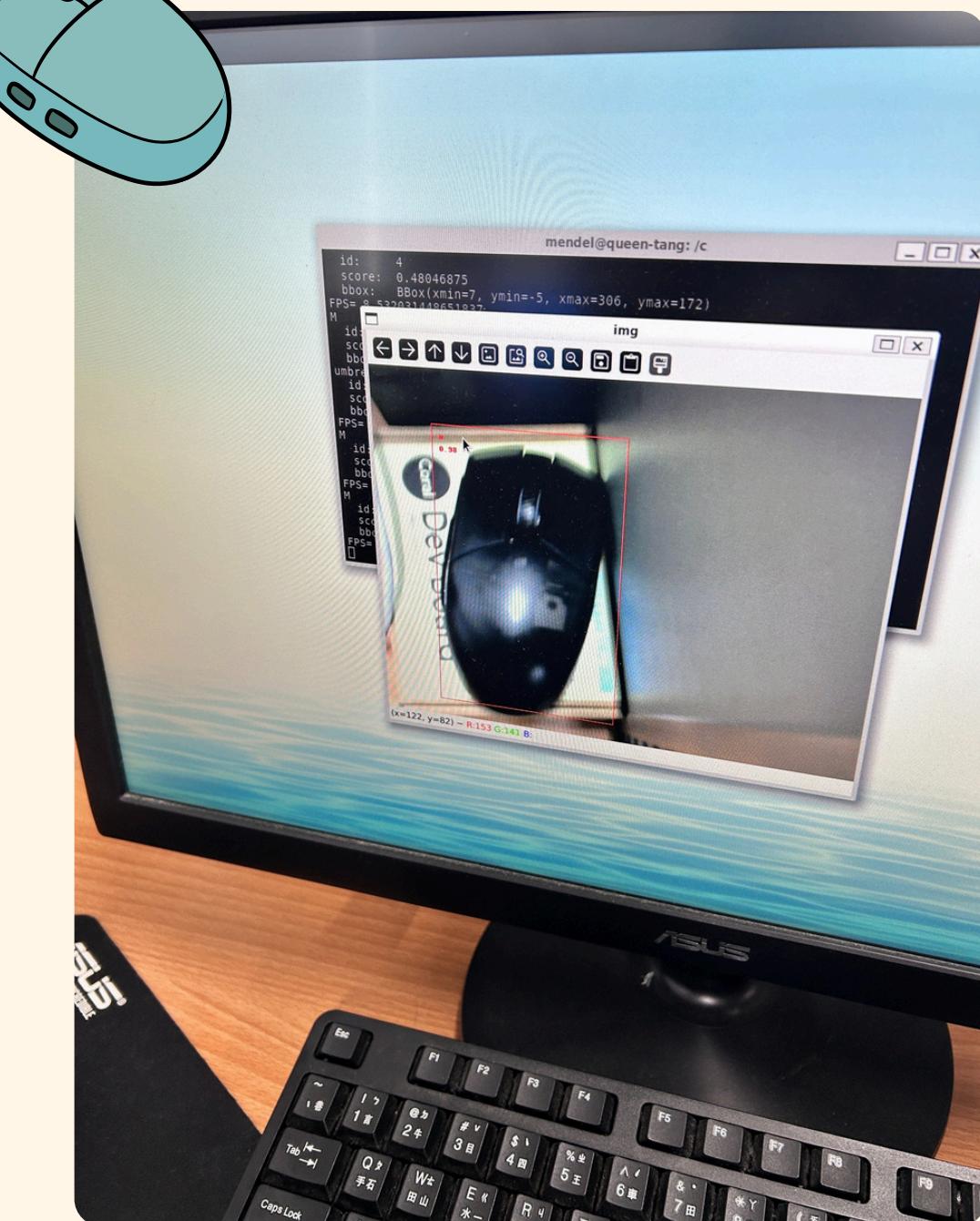
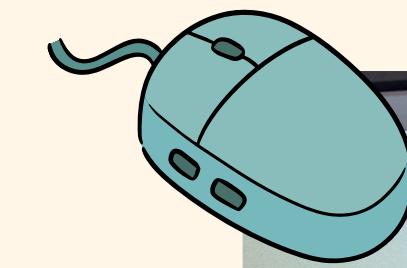
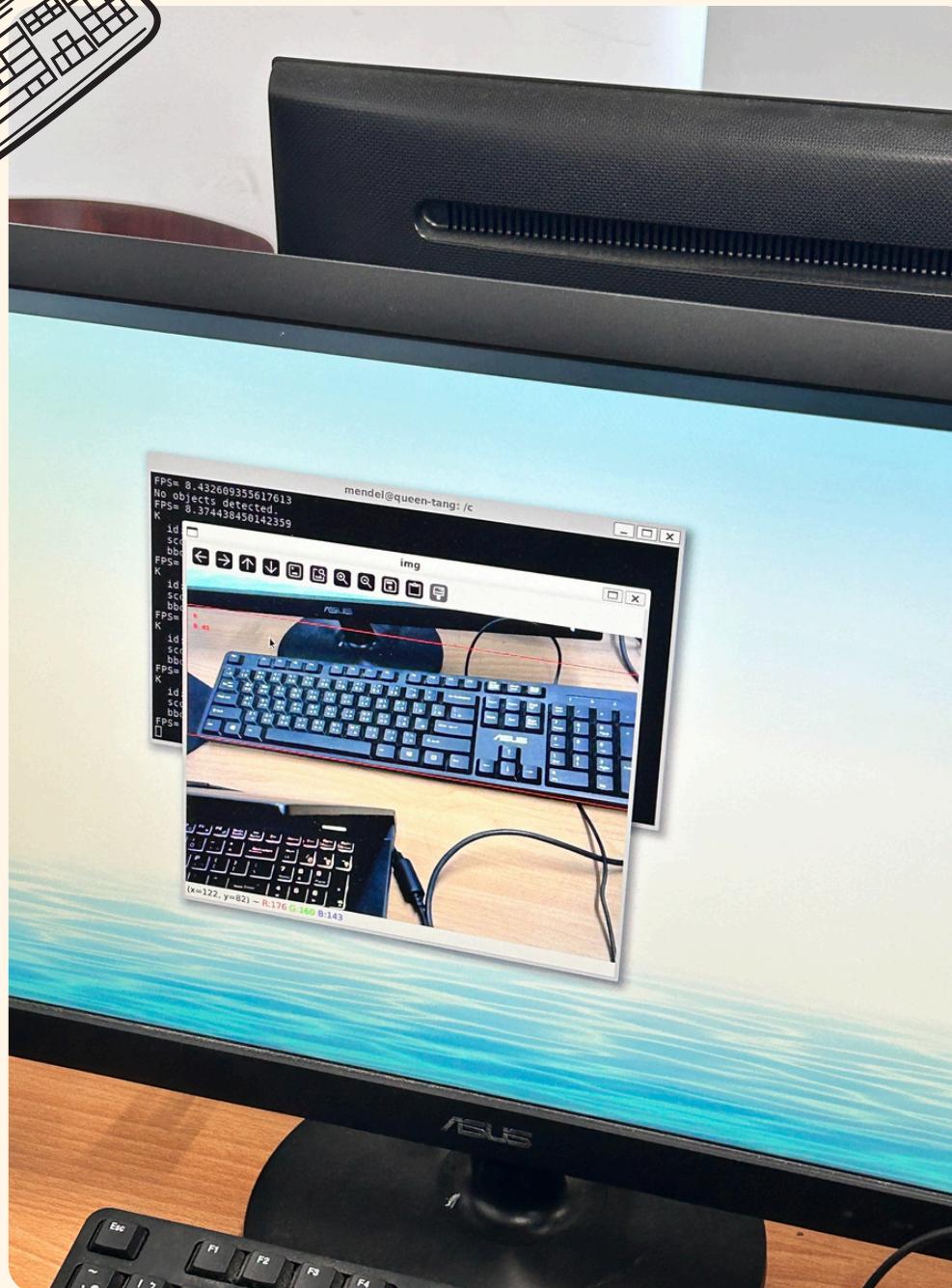
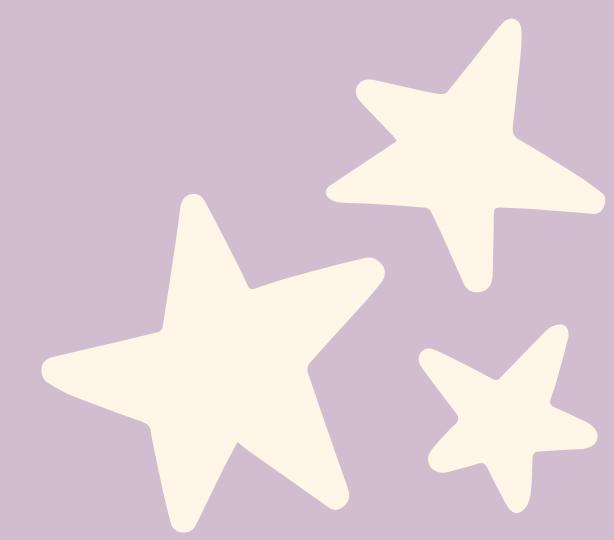
- 掛載USB
- 輸入指令

```
● ● ●  
1 sudo mkdir Lab4 #建立Lab4資料夾  
2 sudo mount./dev/sda1 ./Lab4 #將sda1掛載到Lab4資料夾  
3 cd Lab4 #進入Lab4資料夾  
4 ls #列出Lab4資料夾內容
```

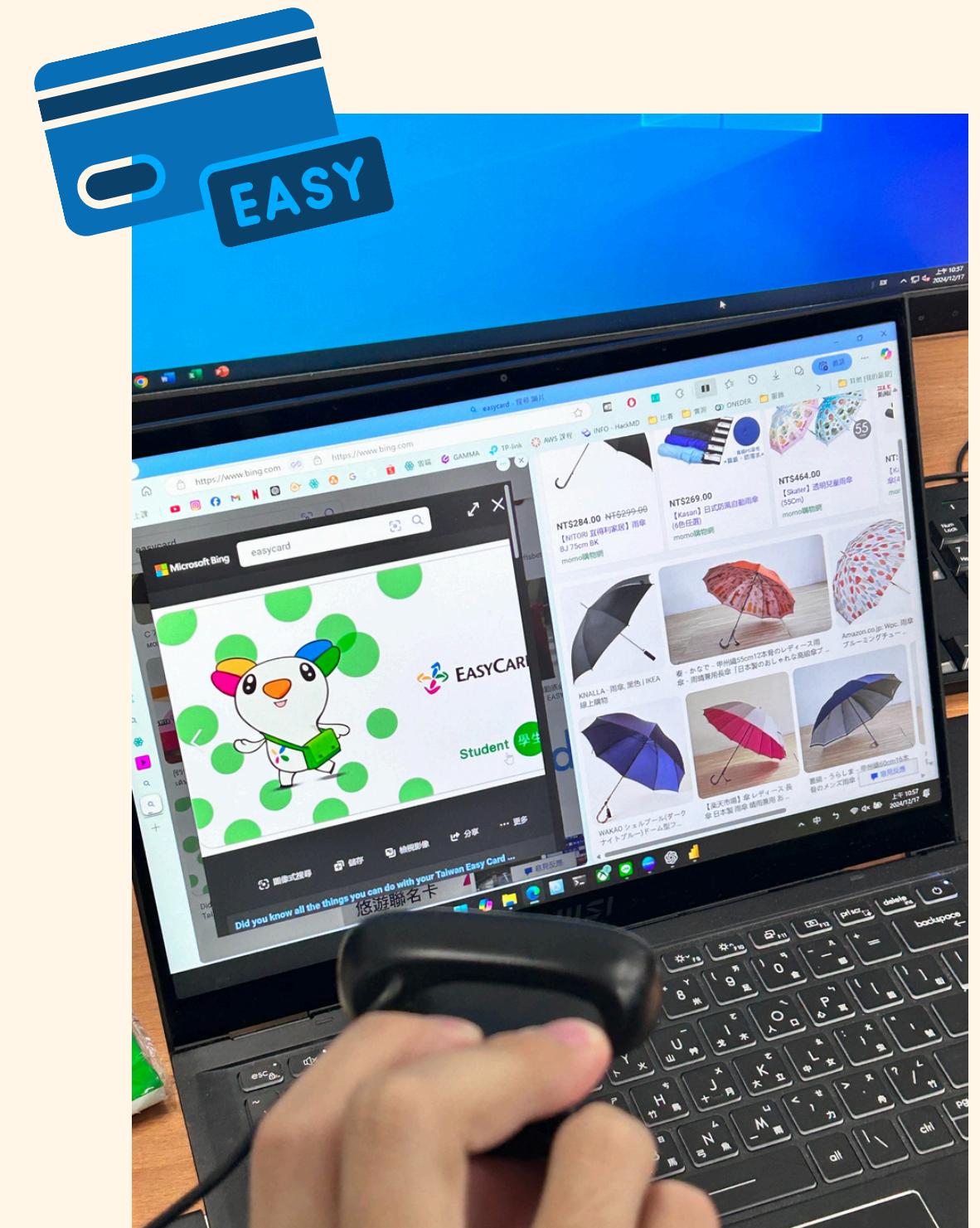
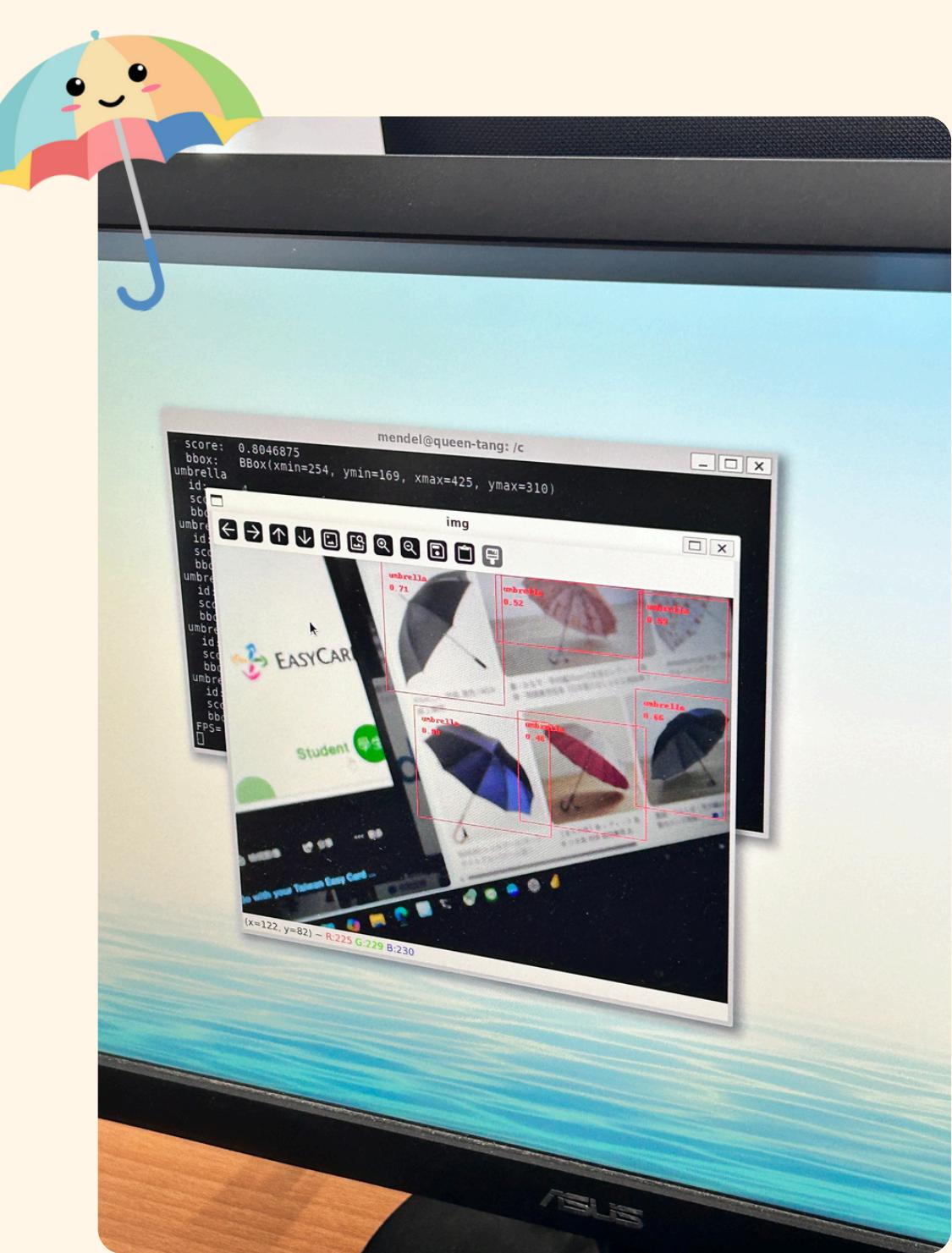
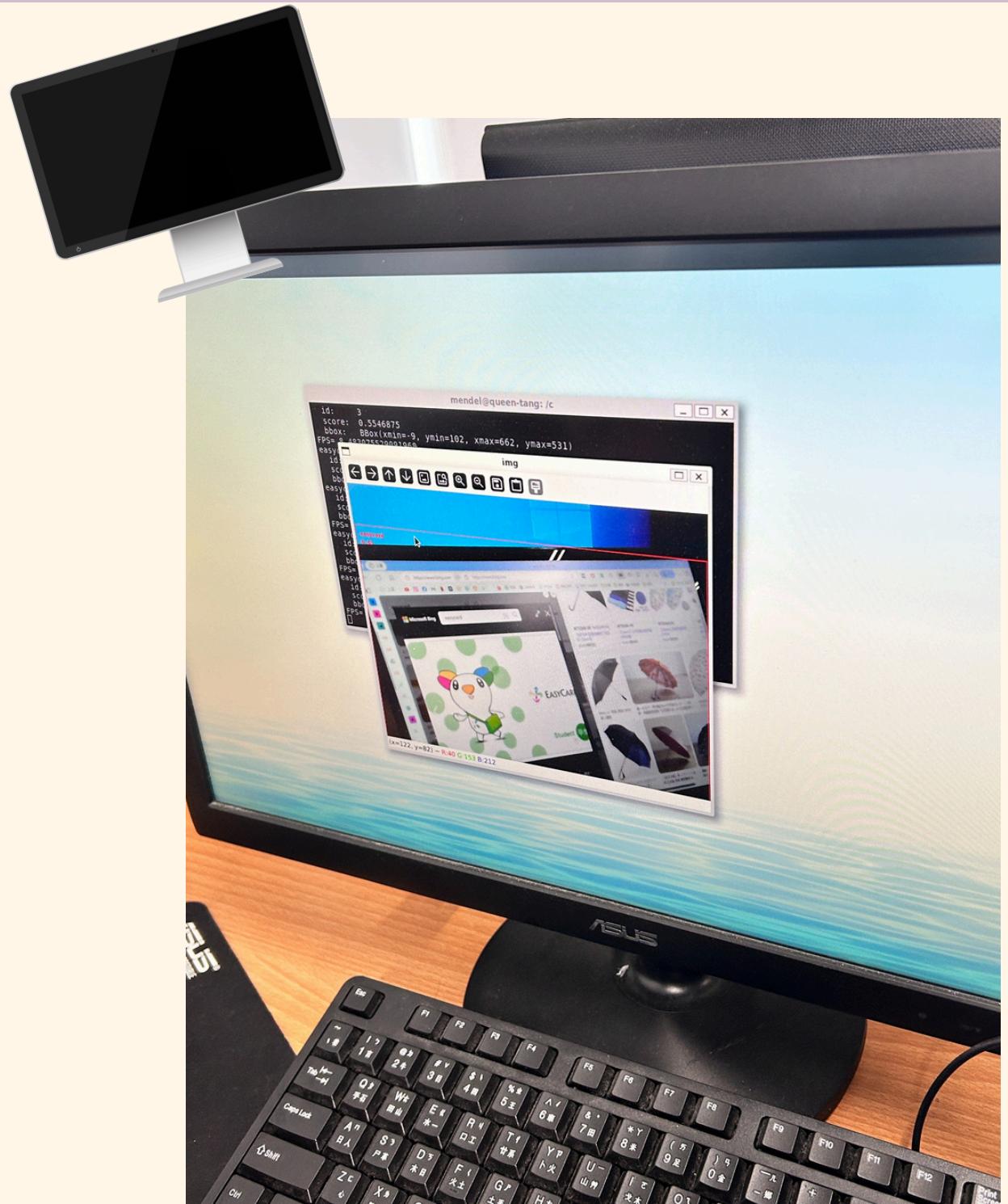
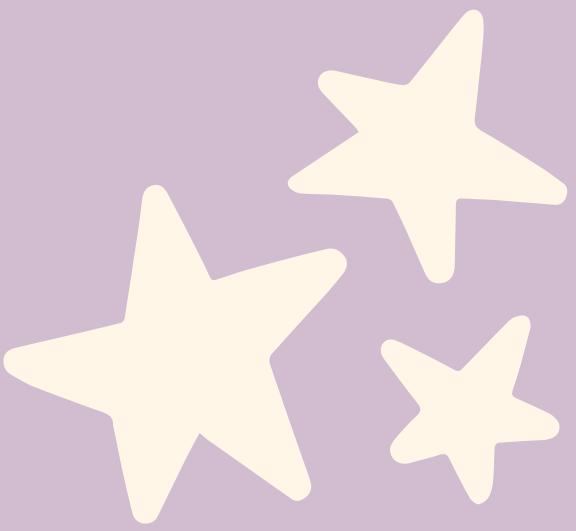
- 在Label.txt增加自己的類別標籤



# 04.EDGE端 驗證結果



# 04. EDGE端 驗證結果



# 05.遇到問題與解決

1

問題：

原始程式碼中設定的訓練參數 epochs=200，訓練次數過多，導致訓練無法在合理時間內完成。

解決方法：

將訓練次數調整為 epochs=80，在保證模型效果的同時減少訓練時間。

```
1 model = object_detector.create(  
2     train_data=train_data,  
3     model_spec=spec,  
4     validation_data=validation_data,  
5     epochs=80,  
6     batch_size=16,  
7     train_whole_model=True,  
8     do_train=True  
9 )
```

2

### 問題：

在新增雨傘相關數據集時，發現部分資料的 XML 標註檔案與 JPG 圖片不一致，導致訓練數據無法正確載入。

### 解決方法：

檢查並修正所有雨傘數據的 XML 檔案與 JPG 圖片的對應關係，確保資料一致性。

```
1 def rename_files(images_path, annotations_path):
2     # 獲取圖片和標註檔案
3     image_files = [f for f in os.listdir(images_path) if f.endswith('.jpg')]
4     annotation_files = [f for f in os.listdir(
5         annotations_path) if f.endswith('.xml')]
6
7     # 確保圖片檔案和標註檔案數量一致
8     if len(image_files) != len(annotation_files):
9         print(
10             f"圖片數量 ({len(image_files)}) 與標註檔案數量 ({len(annotation_files)}) 不一致!")
11         return
12
13     # 根據圖片檔案數量進行重命名
14     for i, (image_file, annot_file) in enumerate(zip(image_files, annotation_files), 1):
15         # 設定新的檔案名稱
16         new_name = f"umbrella_{i:02d}"
17
18         # 重新命名圖片檔案
19         old_image_path = os.path.join(images_path, image_file)
20         new_image_path = os.path.join(images_path, new_name + '.jpg')
21         os.rename(old_image_path, new_image_path)
22
23         # 重新命名標註檔案
24         old_annot_path = os.path.join(annotations_path, annot_file)
25         new_annot_path = os.path.join(annotations_path, new_name + '.xml')
26         os.rename(old_annot_path, new_annot_path)
27
28         print(
29             f"已重新命名: {image_file} -> {new_name}.jpg, {annot_file} -> {new_name}.xml")
```

3

### 問題：

由於訓練次數減少，模型未能充分學習，導致模型容易將悠遊卡的彩色部分誤辨識為雨傘。

### 解決方法：

通過擴充悠遊卡的訓練影像數據集，增加多樣化的圖片，例如不同光線、角度和背景的悠遊卡照片，以提高模型的辨識準確性。



## 06. 心得

在這次專案中，我了解物件偵測模型 EfficientDet 的開發與應用流程，包括資料準備、模型訓練、性能評估以及部署到 EdgeTPU 的實際操作。專案開始時意識到資料質量對於模型效能的重要性，特別是在準備雨傘數據集時，發現部分 XML 標註檔案與 JPG 圖片對應不正確，導致訓練無法正常進行。為了解決這個問題，我逐一檢查並修正所有數據，確保資料的一致性與完整性。這個過程讓我學會了數據清理的重要性，也體會到完善數據前處理是開發流程中至關重要的一環。

在模型訓練過程中，原本設定的訓練參數 (`epochs=200`) 過高，導致訓練時間過長且無法完成。我根據專案需求將訓練次數減少到 80，成功縮短了訓練時間。由於訓練次數的降低，模型在部分情境下出現誤判，例如將悠遊卡的彩色部分辨識為雨傘。所以我重新擴充了悠遊卡的數據集，增加不同光線、角度和背景下的圖片，進一步提升了模型的準確性與泛化能力。這些調整讓我深刻理解了如何根據實際需求動態調整參數並改進模型效能。

模型訓練完成後，我將其轉換為適合 Coral EdgeTPU 的格式，並部署到 Dev Board 上進行測試與驗證。在硬體設備上的推理過程，讓我體會到軟硬體結合的重要性。從模型轉換到實時推理，我學到如何利用硬體特性優化模型的推理速度，並在測試過程中逐步完善系統效能。整個專案不僅加深了我對深度學習完整流程的理解，也讓我在實踐中提升了解決問題的能力，尤其是在數據準備和模型部署方面的細節處理，為未來的專案開發積累了寶貴經驗。

