HW2 研究方法與成果（s1091607）

1. 觀察資料

進行資料處理前，首先必須詳細觀察資料，才能得知資料的特徵與異常之處。觀察資料的過程中，發現 Annotation 資料夾裡的 XML 檔內，每個<*annotation*>標籤都包含了一張圖片的完整資訊。這些資訊包括資料夾名稱、檔案名、圖片的尺寸（寬度、高度、深度），以及缺陷的詳細資訊。而每個缺陷部分都包含在<*object*>標籤內，如缺陷類型（Spurious copper、Spur、Short、Open circuit、Mouse bite、Missing hole，共六種）、缺陷部分在圖片的位置（被包含在<*bndbox*>標籤內，包括 xmin, ymin, xmax, ymax）等。

2. 處理資料

通常模型的網絡架構對輸入圖片的大小都有特定的要求。訓練過程中輸入的圖片大小需要是一致的，因為神經網絡中的卷積層、pooling 層等操作對輸入尺寸有固定設計。另外，統一圖片大小可以提高 GPU 運算的效率，尤其是在使用 batch 處理時。不同的圖片大小可能會導致計算的效率下降。因此要先將所有照片統一大小，這裡使用 Python 的 Pillow 庫（PIL），大小統一為 640，再將調整大小後的圖片統一保存在新的資料夾底下。

調整完圖片大小後，接下來使用 Python 的 xml.etree.ElementTree 模組來解析 XML 檔案，得到圖片的原始尺寸和缺陷部分的邊界框座標，再根據圖片尺寸的變化比例，調整邊界框的 xmin、ymin、xmax、ymax 值，讓邊界框依然可以正確地對應到縮小後的圖片上。如此一來，每個標註的位置都會精確地指向圖片中相應的缺陷部分。再將調整後的 XML 文件存儲到新的文件夾中，檔名的命名方式與縮小後的圖片相同，方便對應。

3.模型選擇

這裡選擇 YOLOv5，YOLOv5 是一個知名的物件偵測系統，而因 YOLOv5s 是 YOLOv5 系列（如 YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x）中最輕量級的版本，其模型大小和運算需求相對較低，使得它不需在特別高性能 GPU 的系統上也能有效運行(後來在 colab 上跑的)，考慮吃效能的問題，選擇 YOLOv5s 模型。

4. 從 XML 轉換到 YOLO 標籤格式

先從 XML tree 中取得每一個<*object*>標籤，這些標籤包含了標記在圖片中缺陷部分的資訊,如缺陷類別(class_name)和邊界框(bbox)座標(xmin, ymin, xmax, ymax)。因為 YOLO 模型仰賴於數字的標註。每個類別（例如 spurious_copper 或 mouse_bite 等）在模型內部都是

透過數字（如 0, 1, 2...）來標註識別的，因此要再使用 class_mapping dictionary 將缺陷的類別名稱映射到對應的索引（如圖 1）。

接著，因 YOLO 訓練需要邊界框的中心點坐標和相對尺寸比例，幫助模型更好學習。為計算中心點和圖片的寬度和高度，取 xmin 和 xmax 的平均，以及 ymin 和 ymax 的平均，算出 x_center 和 y_center，然後除以調整大小後的圖片寬度和高度，將其轉換成相對於圖片尺寸的比例。至於 width 和 height 的計算是透過 xmax - xmin 和 ymax - ymin 得出邊界框的寬度和高度，然後除以調整大小後的圖片寬度和高度轉換成適當比例。最後對於每一個處理過的 XML 檔，創一個同名的.txt 檔，在同一資料夾中寫入轉換後的 YOLO 標籤。

```
class_mapping = {'spurious_copper': 0, 'mouse_bite': 1, 'open_circuit': 2, 'missing_hole': 3, 'spur': 4, 'short': 5}
```

圖 1　class_mapping dictionary

5. Train Set & Validation Set

原始資料集已分為 train、val、test 三種資料夾，將 train set 與 validation set 的資料夾路徑寫入 data.yaml，供模型訓練與驗證評估。（如圖 2）

- train: 包含訓練圖片的資料夾路徑。
- val: 包含驗證圖片的資料夾路徑。
- nc: 代表類別的數量，此例為 6。

- names: 列出所有類別的名稱。

```
data_yaml_content = """

train: /content/PCB_resized/train
val: /content/PCB_resized/val
nc: 6
names: ['spurious_copper', 'mouse_bite', 'open_circuit', 'missing_hole', 'spur', 'short']
"""

with open('/content/yolov5/data/data.yaml', 'w') as f:
    f.write(data_yaml_content)
```

圖 2 data.yaml 內容

6. 訓練模型

為避免記憶體過度負荷，這裡將 batch size 設為 16。嘗試過將 epochs 設為 95、100、110，其準確率分別為 87.6%、89.6%、86.2%，尤其 epochs 設為 110 時就發生 over fitting 了，因此最後選擇準確率較高的 100 為 epochs。從圖 3 中可以看出，在整個訓練過程中，訓練集的 Train Object Loss 持續下降，顯示模型逐漸學習到更精確地分類。

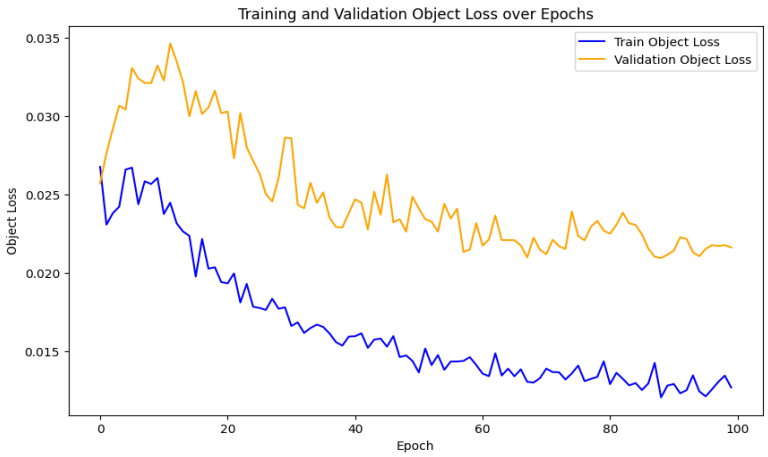

圖 3

7. 測試與成果

最後利用資料集的 test 資料夾中的圖片、XML 檔與 YOLO 標籤 txt 檔（test 的資料皆已處理過），對模型進行測試，以確認其在分類圖片中的各種缺陷上的效果。如圖 4 所示，模型能夠精確地識別缺陷類型。然而，如圖 5 所示，模型在一些情況下未能偵測到所有缺陷，或存在誤判情況。此反映了模型在實際應用中的一些限制，畢竟其整體準確率為 89.6%，尚有提升空間。

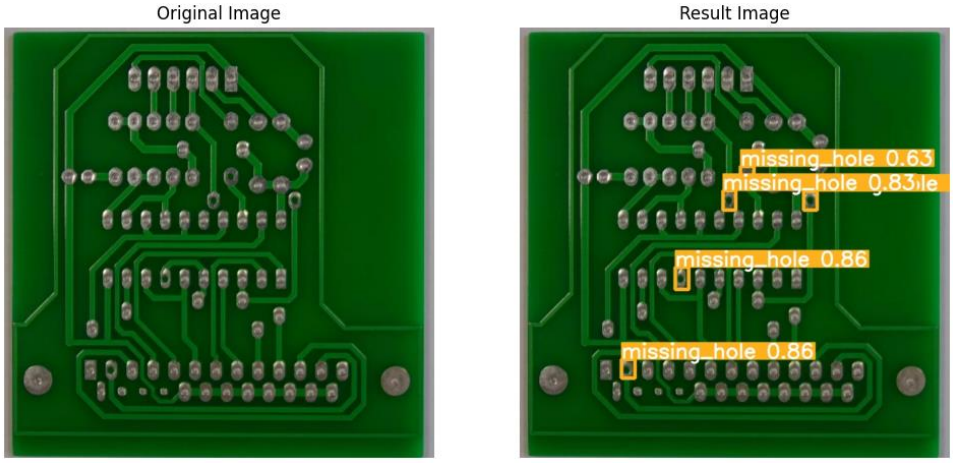

圖 4

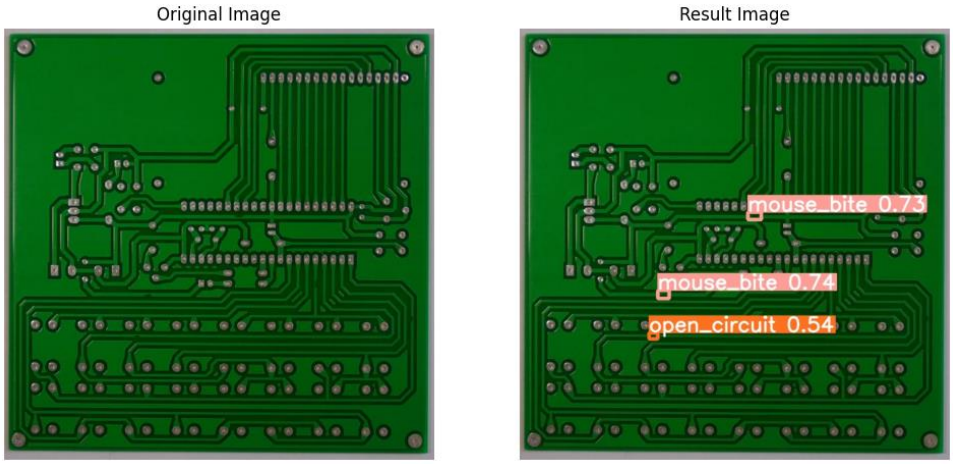

圖 5

圖 6 另存新的 PCB_resized 資料夾之結構

```
100 epochs completed in 0.409 hours.
Optimizer stripped from /content/yolov5/runs/HW2/weights/last.pt, 14.5MB
Optimizer stripped from /content/yolov5/runs/HW2/weights/best.pt, 14.5MB

Validating /content/yolov5/runs/HW2/weights/best.pt...
Fusing layers...
YOLOv5s summary: 157 layers, 7026307 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 3/3 [00:04<00:00,  1.37s/it]
                   all         92        467      0.896      0.736      0.828      0.369
       spurious_copper         92         82      0.982      0.673      0.862      0.426
            mouse_bite         92         76      0.963      0.676      0.795      0.328
          open_circuit         92         81      0.953      0.679      0.865      0.403
          missing_hole         92         78      0.961      0.974      0.979      0.492
                   spur         92         75      0.623      0.627      0.636      0.261
                 short         92         75      0.894      0.789      0.832      0.302
Results saved to /content/yolov5/runs/HW2
```

圖 7 訓練結果