Homework 2

資訊所 P76134082 陳冠言

Project: scaphoid and fracture detection

目錄

（一）專案簡介

專案目標

使用 X 光的影像分析是否有骨裂（Fracture）的形況，若有骨裂則顯示骨裂位置，若無則不須標註，最後要產生預測的 Accuracy, Precision, Recall 以及 Bounding Box 的 Mean IOU。

專案方法

由於圖像骨裂位置佔整體面積較小，因此若直接預測效果會不好（已實驗過），因此本專案先透過 Yolov8l 進行 Segmentation，框出舟骨位置（Mean IOU＝0.92），接著針對舟骨位置進行 Fracture Detection，使用的模型一樣是 Yolov8l。

實驗結果

本專案以 80%的資料作為 Train data、20%的資料為 Validation data，並將全部的資料作為 test data 來進行 Inference，最後得到評估結果如表 1 和圖 1 所示。

表 1 - Evaluation

Accuracy	Precision	Recall	Mean IOU
0.9	1.0	0.7959	0.6704

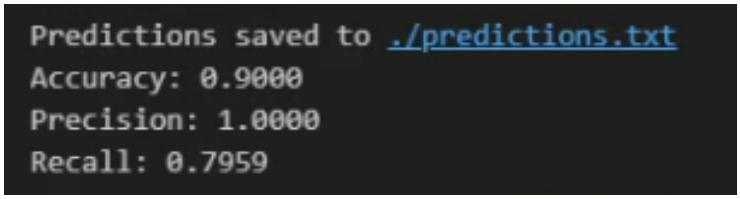

圖 1 – 程式碼輸出證明

UI 介面

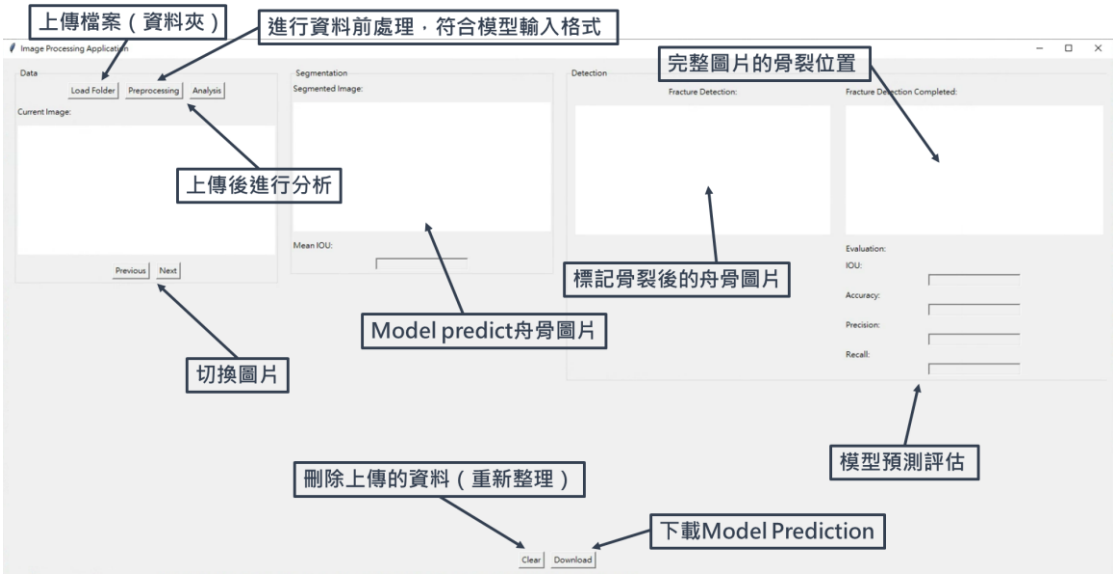

圖 2 – 系統 UI 以及功能介紹

（二）資料前處理

原資料格式

資料分成 scaphoid_detection 和 fracture_detection 的資料夾。

1. scaphoid_detection 資料夾中有 images 和 annotations 兩個資料夾，images 存放所有影像資料，而 annotations 存放對應檔案名稱的 json 檔案，並且有 bounding box 的座標（左上角座標 x1,y1 和右下角座標 x2,y2）。
2. fracture_detection 資料夾中有 annotations 資料夾，裡面存放的 json 檔案對應上述提到的 images 資料的檔案名稱，並且裡面有 fracture 的有無資訊以及 bounding box 的座標（左上、左下、右上、右下四點座標），且座標是以 scaphoid_detection 的座標裁切後的圖片為單位。

scaphoid_detection & fracture_detection 的前處理

1. 讀取 json 的檔案格式與內容，並將 bbox 的座標紀錄後對應同檔案名稱的 image。
2. 將資料封裝成 yolo 可讀取的格式，如：class_id x1 y1 x2 y2 x3 y3 x4 y4。
3. 切分資料夾將 80%作為 train、20%為 validation。
4. 寫下 data.yaml 讓 yolo model 可以讀取。

Data Augmentation

另外，除了原本的資料，我也有做 data augmentation 透過影像翻轉、隨機裁剪等方式處理，因為 scaphoid_detection 的效果已經不錯了，因此這部份只有針對 fracture_detection 做，產生出來的圖片如下所示。

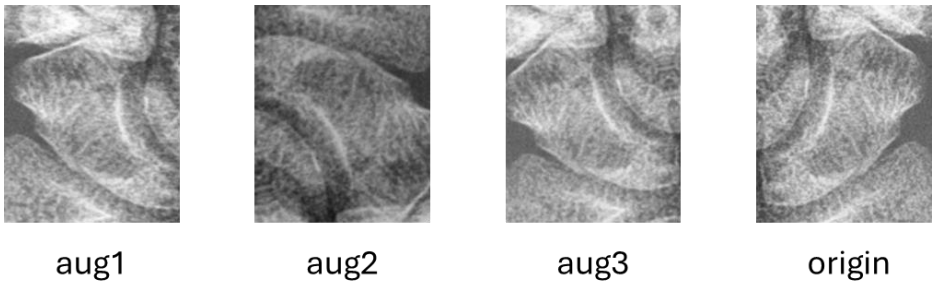

圖 2 – Data Augmentation 範例

scaphoid_detection & fracture_detection 模型選擇

本專案使用 Yolov8-OBB 作為 backbone model，因為一般的 yolo 無法支援有角度的 bounding box，而 OBB 可透過給予四個頂點的 XY 座標來計算角度。

圖 3 – Oriented Bounding Boxes Object Detection Example

（三）舟骨偵測（scaphoid detection）

實驗過程

針對舟骨偵測的任務，我嘗試過許多模型和架構來實作。

1. 使用 FastRCNN 實作，實作效果還可以，但在 fracture detection 的表現很差所以後來就不採用為最終 model。
2. 使用 yolov5 實作，效果良好，同時因為舟骨偵測的 bounding box 沒有角度問題，因此這個 model 就已經有很好的表現，但因為 fracture detection 是使用 yolov8l 作為初始化權重的模型，考量到統一性，因此最終就使用 yolov8 來作為 backbone。
3. 使用 yolov8l 來實作（最終 model），表現跟 yolov5 差不多。

實驗結果

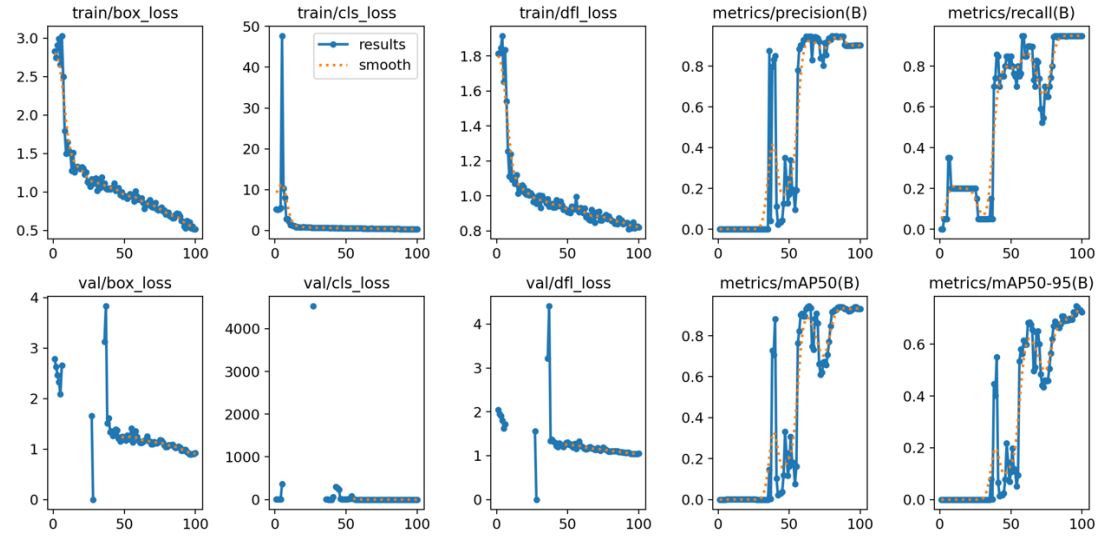

圖 4 – 訓練歷程圖

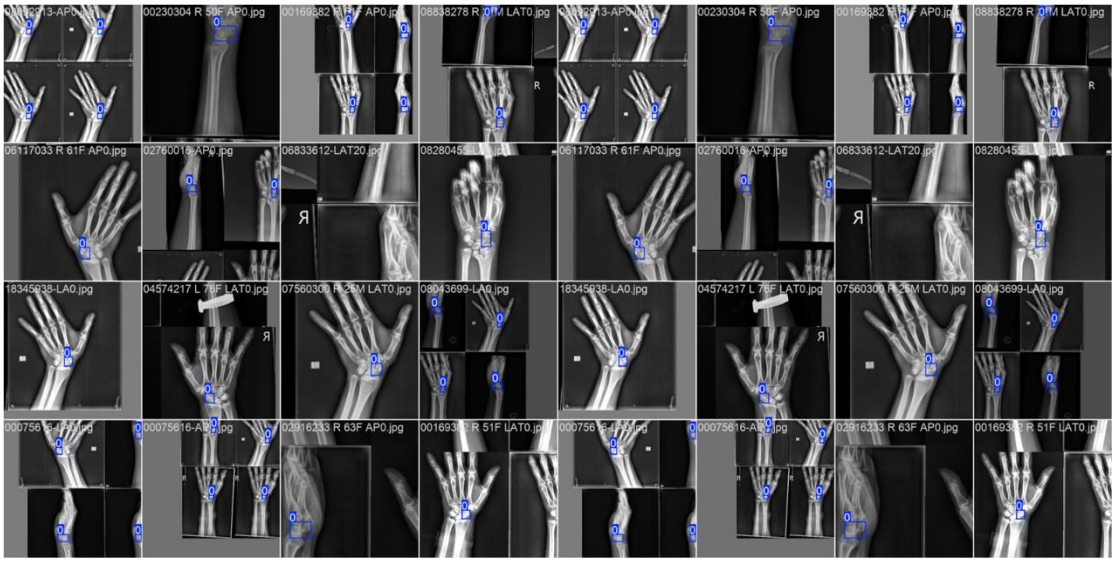

圖 5 – batch prediction

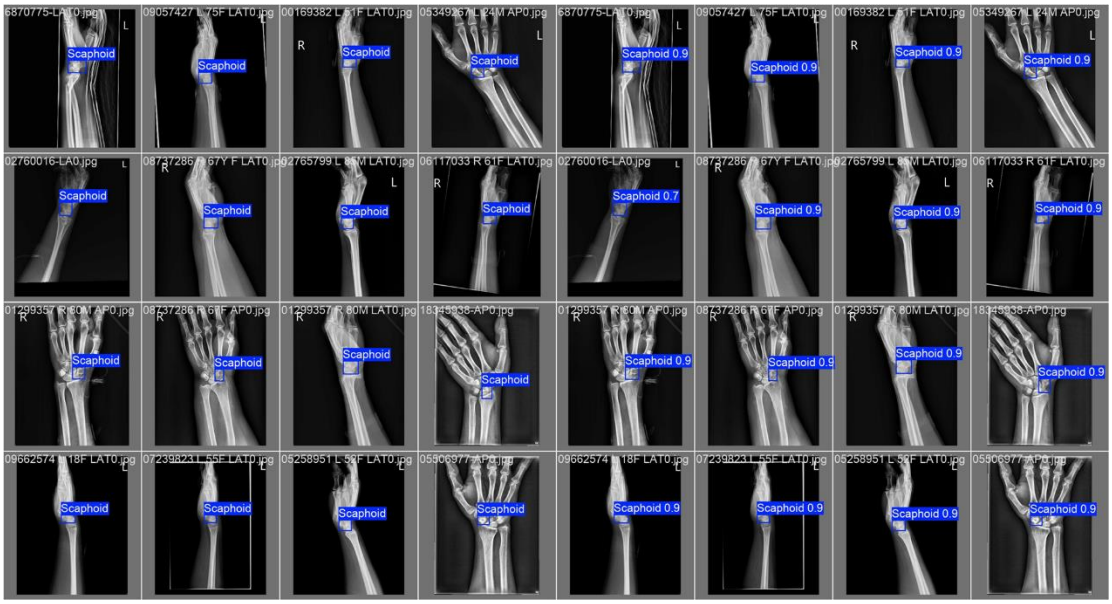

圖 6 – validation prediction（labels/bounding box）

（四）骨裂偵測（fracture detection）

實驗過程

相較於舟骨偵測，骨裂偵測的難度較高，不僅目標較小難以偵測，同時 bounding box 也存在著不同角度，若使用一般的 yolo 模型是無法有良好表現的，因此我也實驗過許多不同的模型。

1. 使用 yolov5 實作，效果很差且 bounding box 很大角度也不對。
2. 使用 rotation-yolo 實作，效果不佳，訓練此模型時由於該模型版本太舊，numpy 或其他的套件版本已不支援許多 function，因此主要都在 debug，不知道是因為資料處理的過程格式有出入還是內部 function 有些沒調整好而訓練好的效果很差，後來就先轉而嘗試 yolo obb 了。

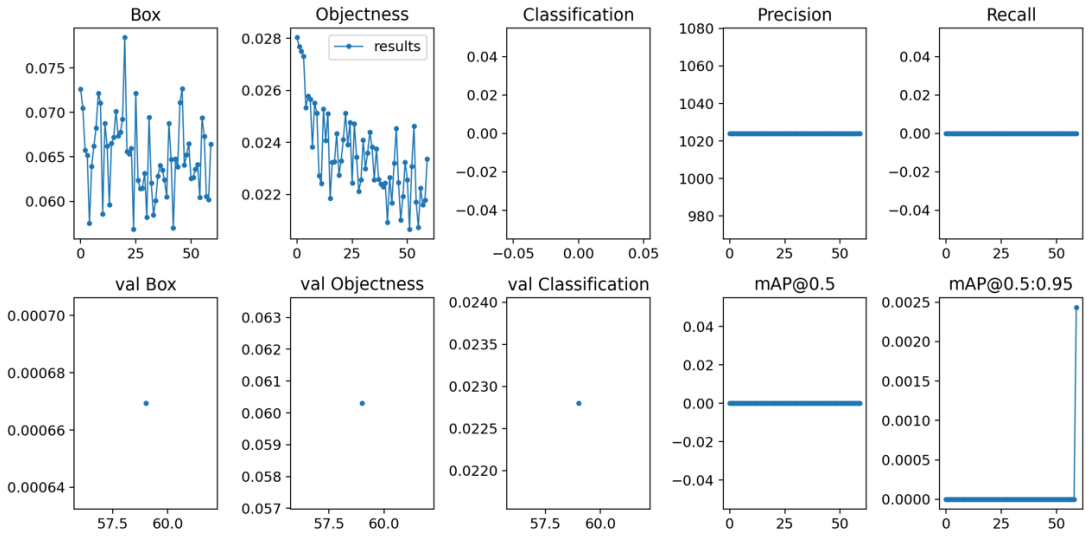

圖 7 – rotation yolo results

3. 使用 yolo-obb 實作，與 rotation yolo 不同，obb 可以支援四個頂點的 xy 座標或 x 和 y 中心點座標搭配角度的資訊作為輸入，而 rotation yolo 只能使用後者，且角度上也有自己的規格，而 obb 的訓練結果表現良好也因此作為我最後採用的模型，而 backbone 的部分我有使用過 yolov8m 和 yolov8l 來做，兩者差異不大，但 yolov8l 的 parameter 較大表現有好一點。
4. 另外我也有實現直接針對完整圖片進行 fracture detection，但 yolo 模型會直接框出整個手臂或是 fracture 的 bounding box 很大的情況，因此後來就轉回來先做 segmentation 再 detection。

實驗結果

以下是使用 yolov8-obb model 進行 fracuture detection 的一些實驗結果：

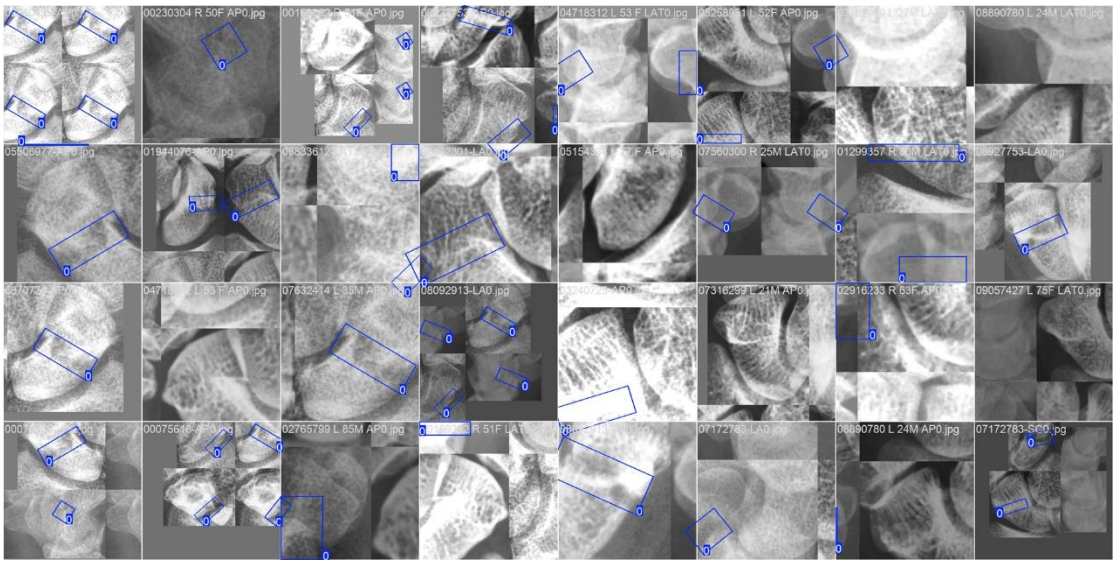

圖 8 – batch prediction

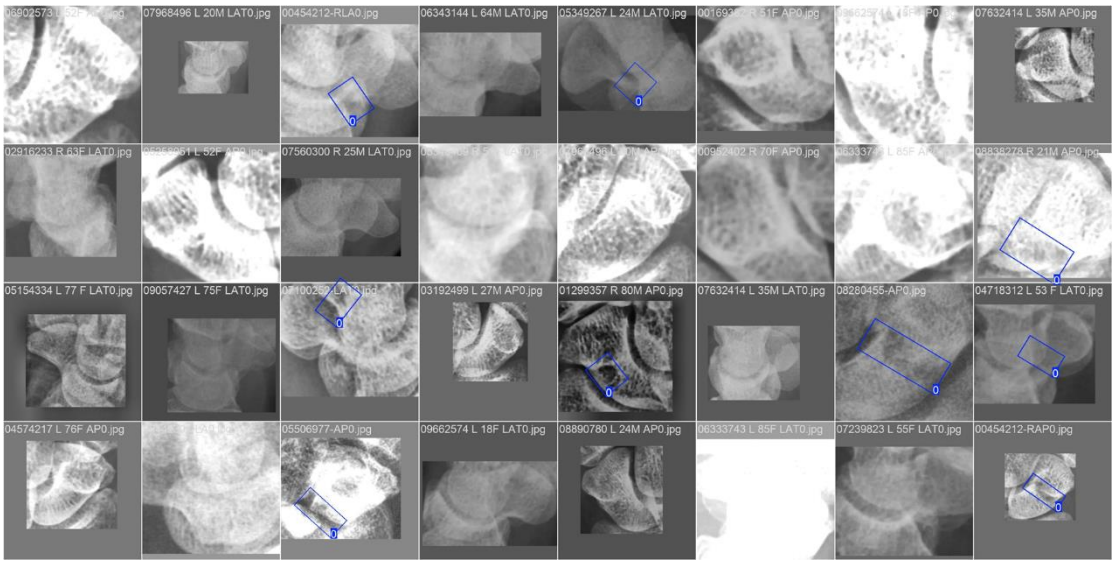

圖 9 – batch prediction 2

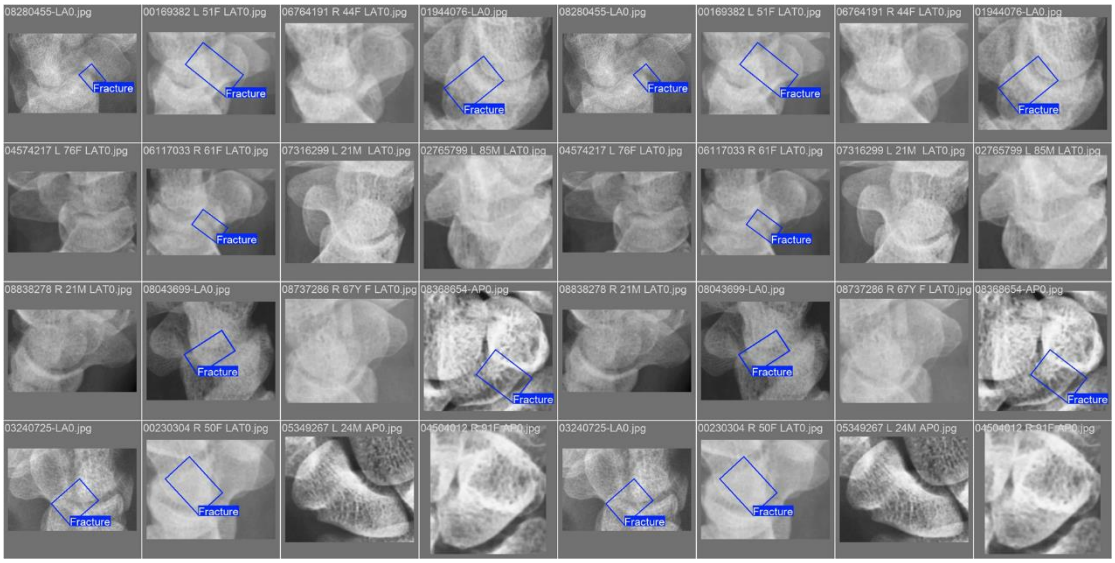

圖 10 – validation prediction

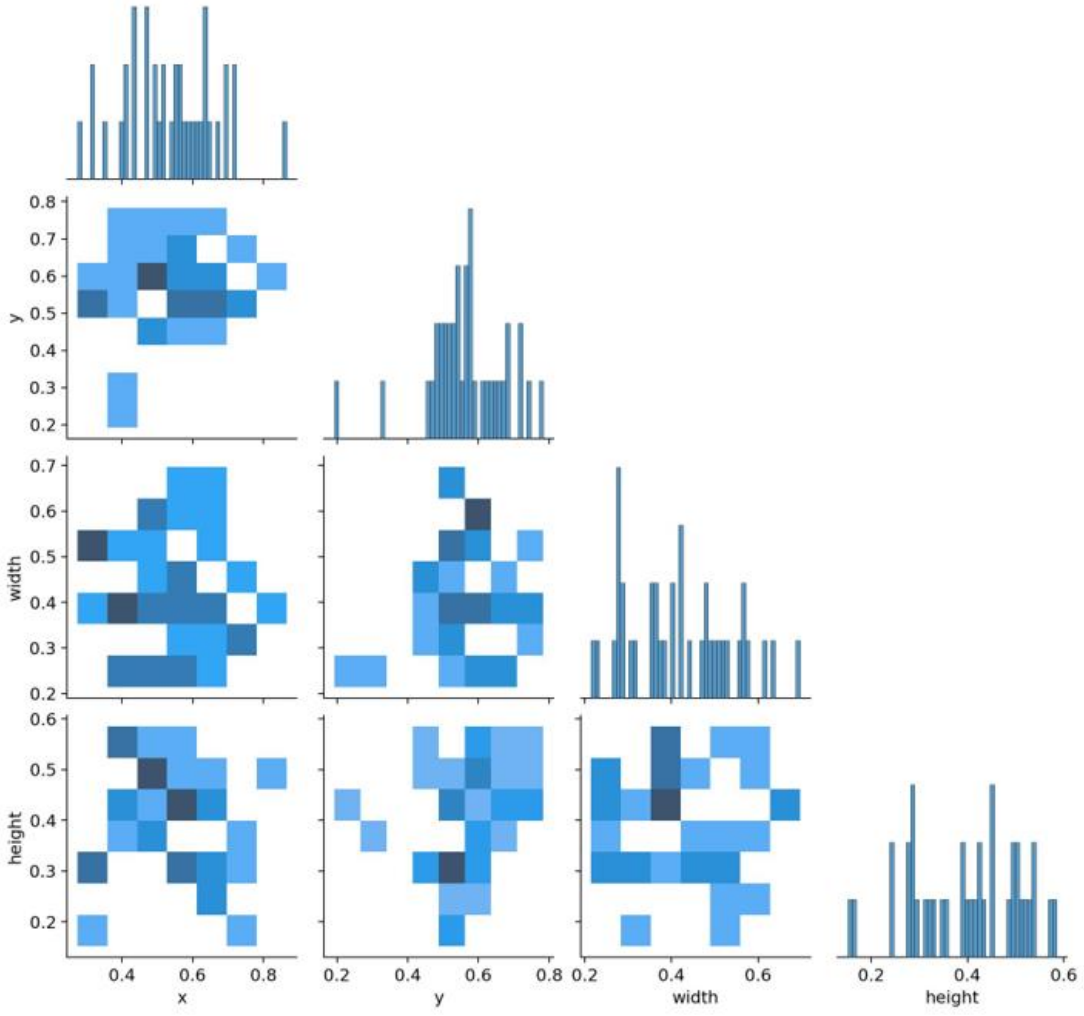

圖 11 – labels_correlogram

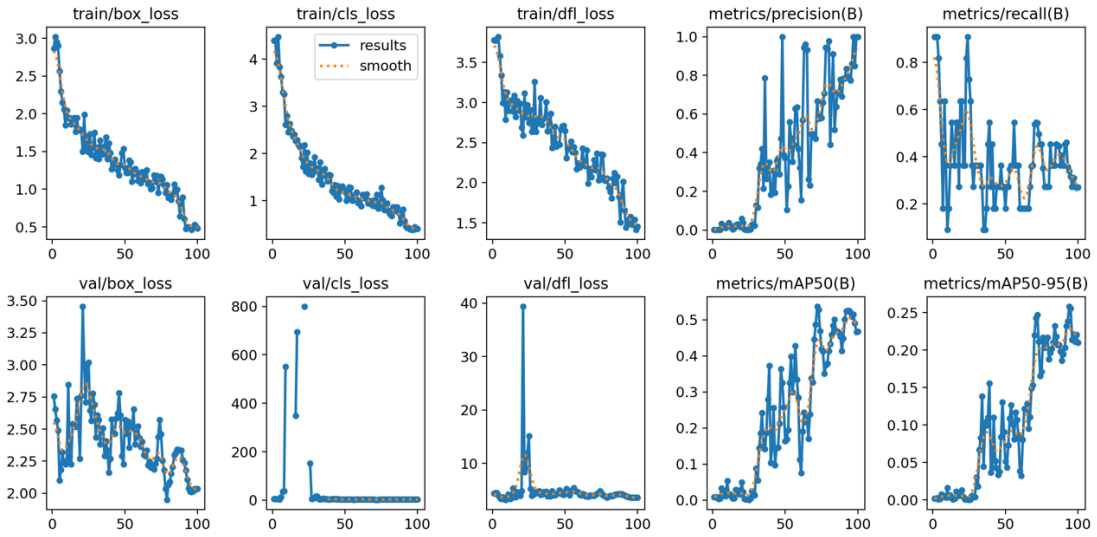

圖 12 – yolov8l 訓練歷程圖

Evaluation

由於我的資料主要為 80% training data 和 20% validation data，因此我使用全部的 data 進行 inference，並且 fracture detection 的部分需要先將 scaphoid detection 的座標記錄下來後，再把 model 預測的 bounding box 座標加上去，而我的做法就是在 scaphoid detection 的部分會先記錄下 segmentation 後最左上角的 bounding box 座標，並將 fracture detection 的四點座標加上此座標後放回原本的完整圖片中，而成品如下圖所示，藍色是 prediction，紅色是 ground true。

圖 13 – model prediction and ground true

（五）應用程式 UI

應用介面是使用 tkinter 套件來製作，而設計理念為根據使用者的使用流程來設計，使用者會先上傳資料後再按下 analysis 來進行 model inference，整體 inference time 大約為 30 秒，而初始畫面如圖 14 所示。

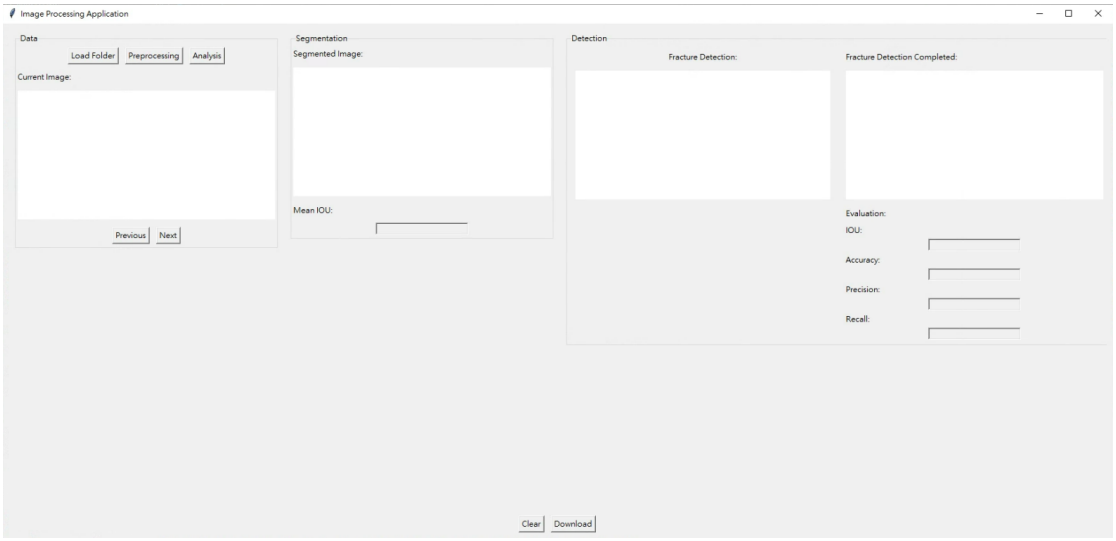

圖 14 – UI 介面

接著畫面會顯示原始圖片、舟骨的 segmentation 圖片、segmentation 並 annotation 的圖片以及 fracture detection 的圖片，如圖 15 所示。

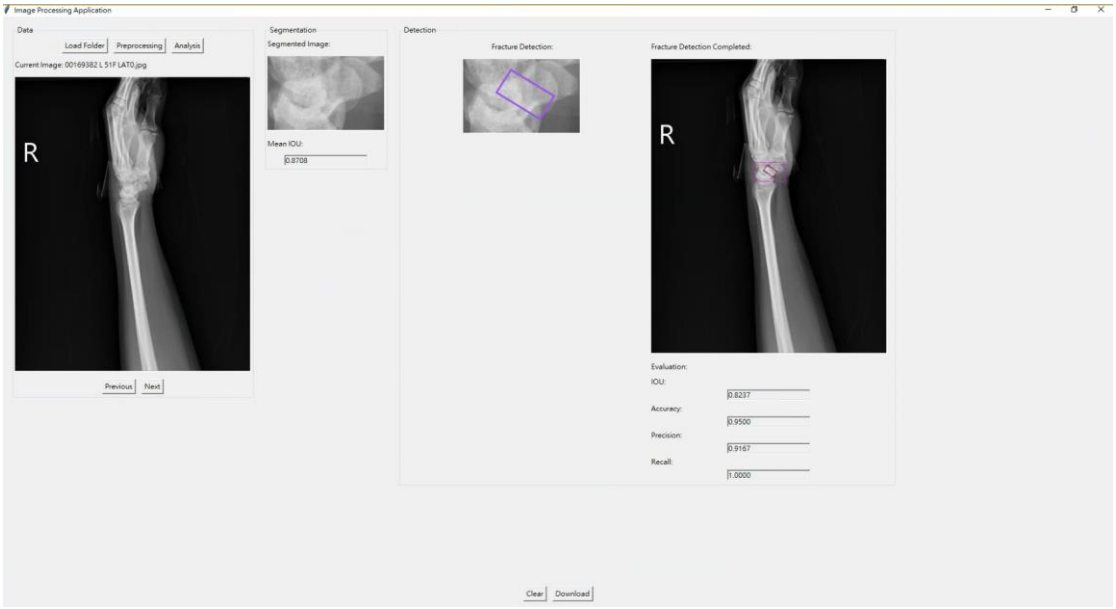

圖 15 – UI 分析後的介面

並且圖片可以透過 previous 和 next 的按鈕進行切換，查看不同圖片的分析效果與結果。接著使用者可能會想進行別的資料 inference 或是想把本次分析結果的資料下載，這時就可以按下 clear 或 download，分別用來重新整個系統和下載本次分析結果資料的功能。另外，本系統其他的預測結果如下圖展示。

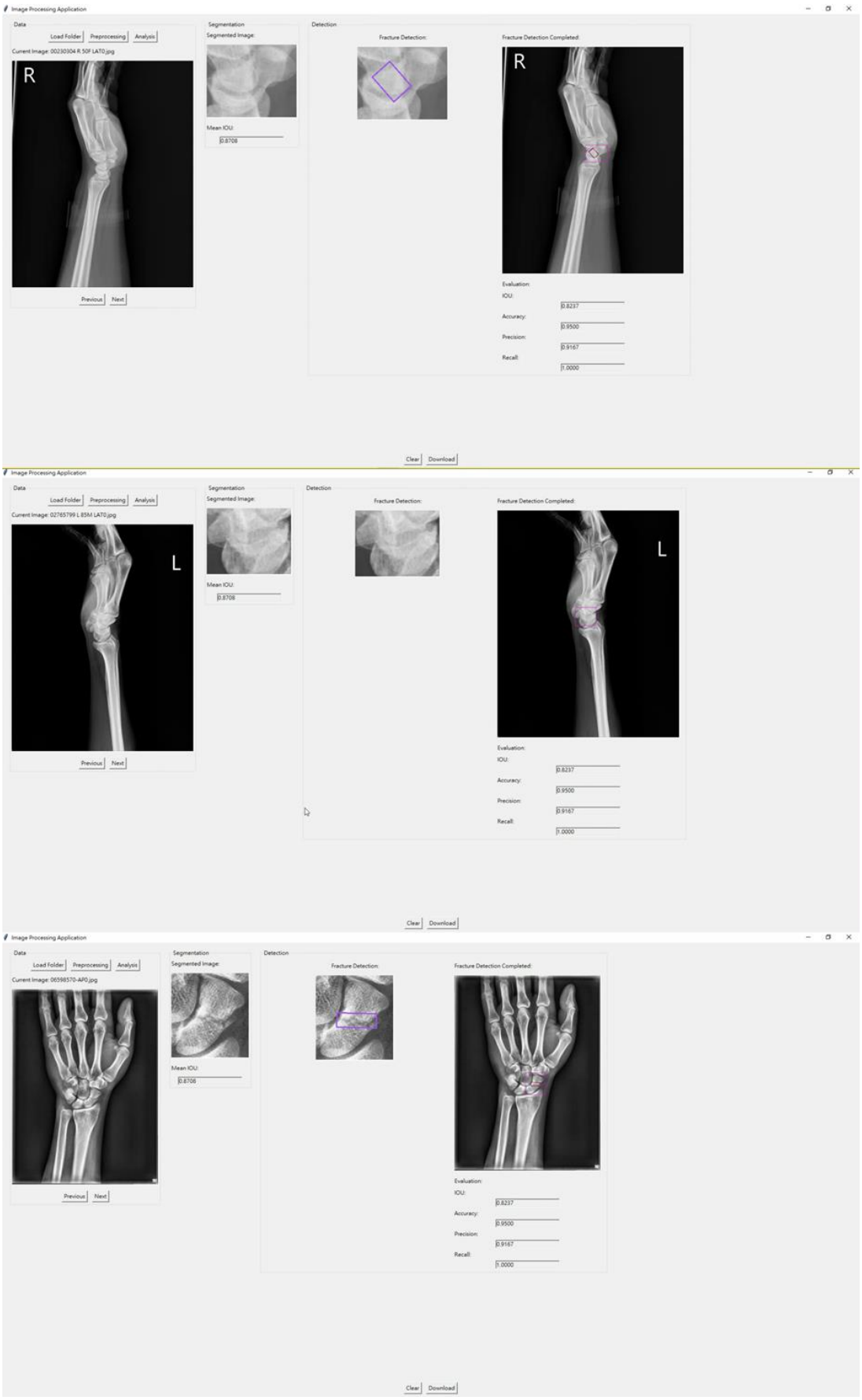

圖 16 – 其他分析結果（含 fracture 及無 fracture）

（六）Conclusion

透過本次作業我也學到許多物件偵測的相關技術，原本以為只需要呼叫 pretrain model 就可以直接解決，但實際上還需要做許多資料前處理包含符合模型的輸入格式、資料增生等步驟，同時在分析後也需要去比對與 ground true 的差異和 bounding box 的計算方式，而我計算 IOU 的方式是透過四個頂點去計算的，主要是透過 Polygon 這個 function，分別建立 Ground Truth 和預測框的多邊形，並利用 polyA.intersection(polyB).area 這個 function 去計算兩個多邊形的交集面積，最後使用 polyA.union(polyB).area 計算兩個多邊形的聯集面積以回傳交集面積與聯集面積的比值，如圖 17 所示。

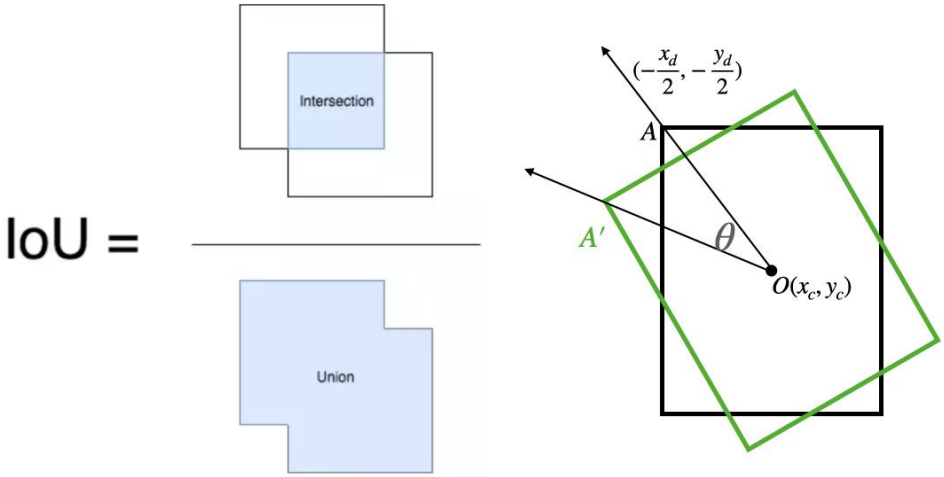

圖 17 – IOU 計算

另外 Data Augmentation 的部分，我使用 validation 的資料進行實驗，透過我的實驗結果發現，有 augmented 的分類效果和 IOU 值皆會下降，且分類效果較不好，兩者結果如表 2 所示。但因為後續助教又有增加其他資料量，因此我再拿新的資料來做測試，結果發現有進行資料擴增的 performance 較好，所以可推論 Data Augmentation 可以增加模型的 robustness，對於沒看過的資料有更好的表現，泛化能力較強。

表 2 – 有無 Data Augmentation 的比較

Model	Accuracy	Precision	Recall	Mean IOU
Origin Model	0.9100	1.0000	0.8200	0.6624
Augmented	0.8700	0.9744	0.7600	0.5679

Reference

https://blog.csdn.net/qq_40672115/article/details/135713830
https://docs.ultralytics.com/modes/predict/
https://supervision.roboflow.com/annotators/
https://docs.ultralytics.com/models/yolov8/
https://github.com/ultralytics/ultralytics