專題報告

資訊110甲
作者:F84064014 朱柏綸
指導教授: 謝孫源

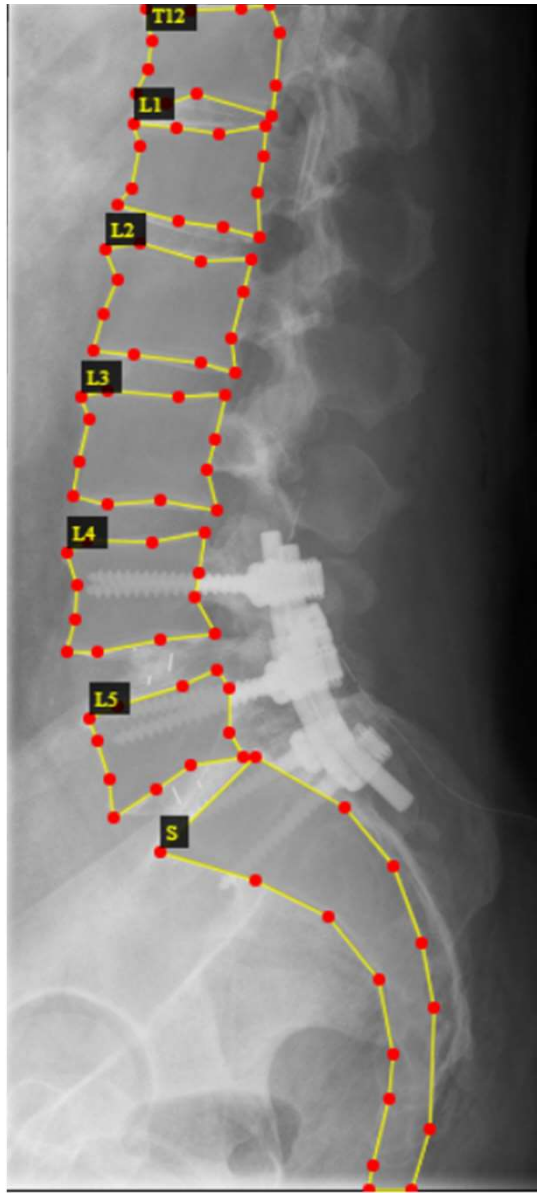
Vertebral Fracture Detection

- 題目: 基於機器學習的X光片 骨折偵測
- 目標: 可以利用深度學習與機器學習方法在X光片 中找出有發生骨折的脊椎
- 主要作業內容:
  1. 資料標記
  2. 脊椎位置偵測
  3. 脊椎鋼釘偵測
  4. 脊椎邊界偵測
  5. 脊椎狀況判斷
  6. 視覺化與執行

資料標記

- 人工標記資料作為Ground Truth，越多資料標記深度學習的模型訓練效果愈好。

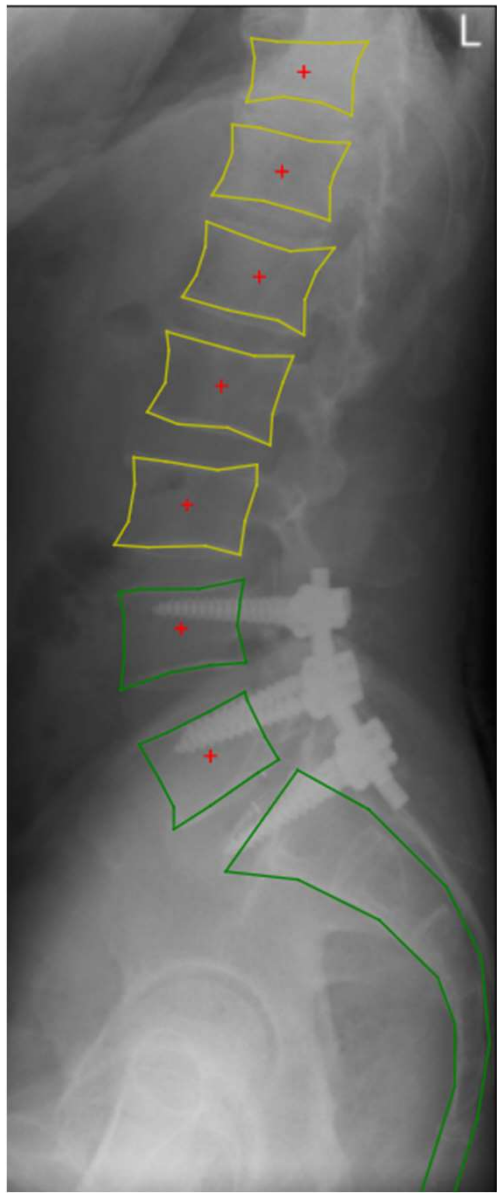
- 目標
  1. 脊椎外型: 每截以12個點大略框出
  2. 脊椎位置: 標記錐體的確切位置(S:薦椎, L1:5腰椎, T1:12胸椎)
  3. 脊椎狀況: 標記脊椎的狀況(normal:正常, compre: 壓迫性骨折, burst: 爆裂性骨折, unsure: 鋼釘)



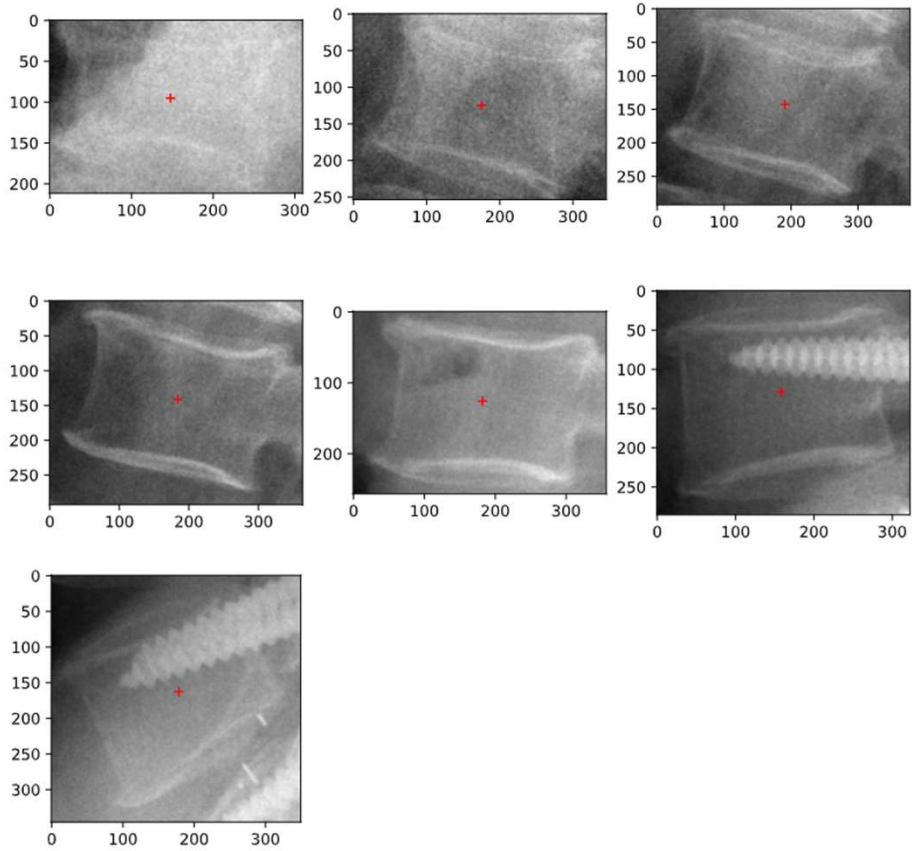
資料標記

- 根據標記資料產生的標準json格式將資料載入格式作圖

- 總計
  1. 共143張圖
  2. 共1486截脊椎



資料標記



脊椎位置偵測

- 目標: 當給定一張未標記過的脊椎X光片時，可以正確得找出所有脊椎的中心點

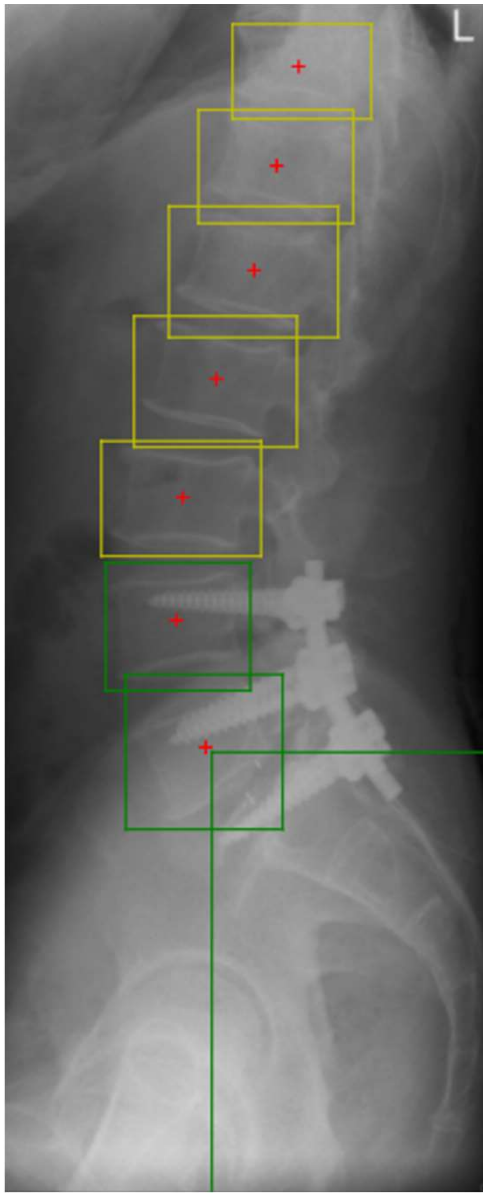
- 作業內容:
  1. 前處理
  2. 模型訓練
  3. 測試結果

脊椎位置偵測 - 前處理

- 前處理

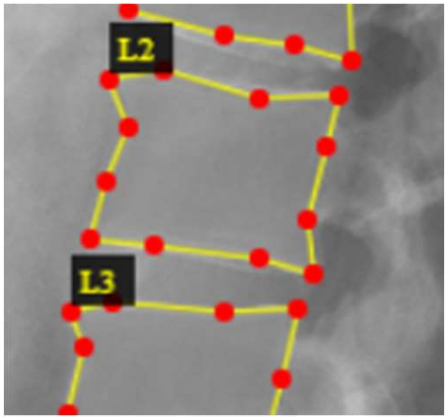
1. 將資料載入程式做train test split，分為train set與validation set，使用train/valid 比例 9:1(以image切割，e.g. 100張圖片 ，90張做為training set 10張做為validation set，由於每一張X光片的脊椎個數不一樣，因此label的比例未必為9:1)

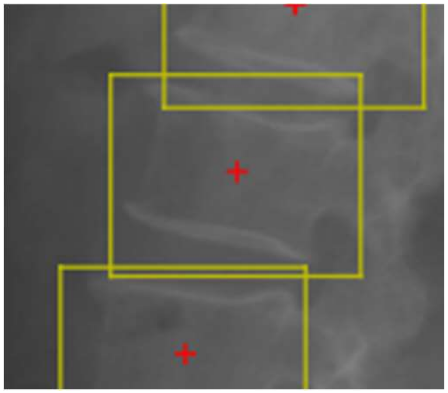
2. 將標記資料格式輸出為yolov5(使用的模型)可以接受的格式(右圖)



脊椎位置偵測－前處理

- 在原資料中找出每一截脊椎在x座標的最大與最小值與y座標的最大與最小值
- 利用所得到的xmax, xmin, ymax, ymin描繪出yolov5形式的方框
- Yolov5格式: [class, x_center, y_center, width, height]，且均須做normalize



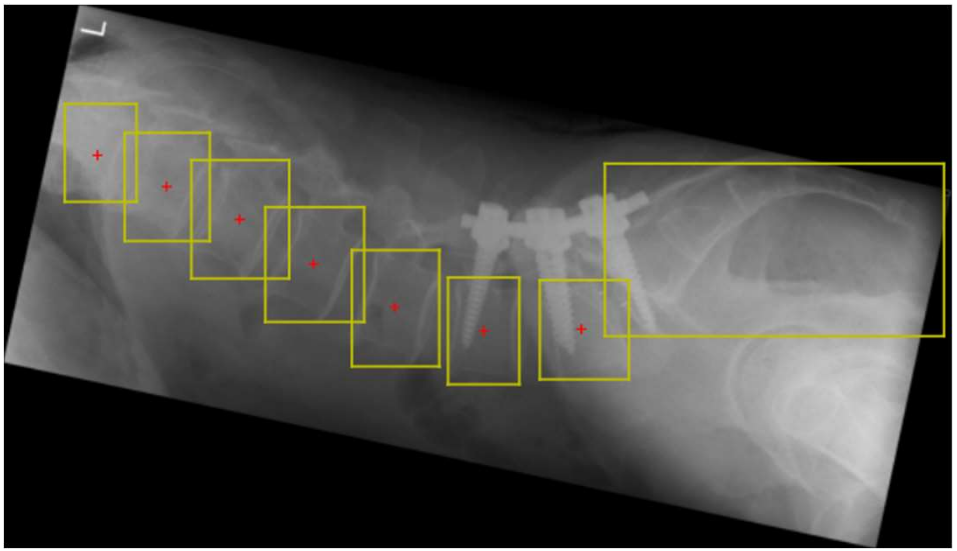


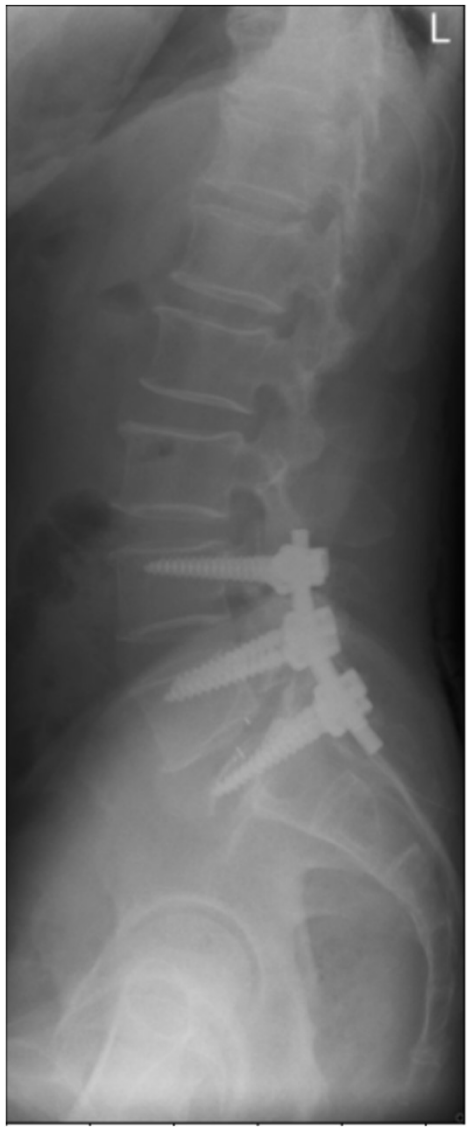
脊椎位置偵測-前處理

- 由於醫療上的需求，由時候需要彎腰的X光片
- 彎腰的資料相較之下辨識度可能較正常的資料低落
- 為了增加模型對此類型資料的辨識度，將一些旋轉過的資料擴充至train dataset

脊椎位置偵測-前處理

- 將原本的脊椎(右圖)隨機選轉(逆時針90到-30度)加入為新的train dataset，結果如下圖

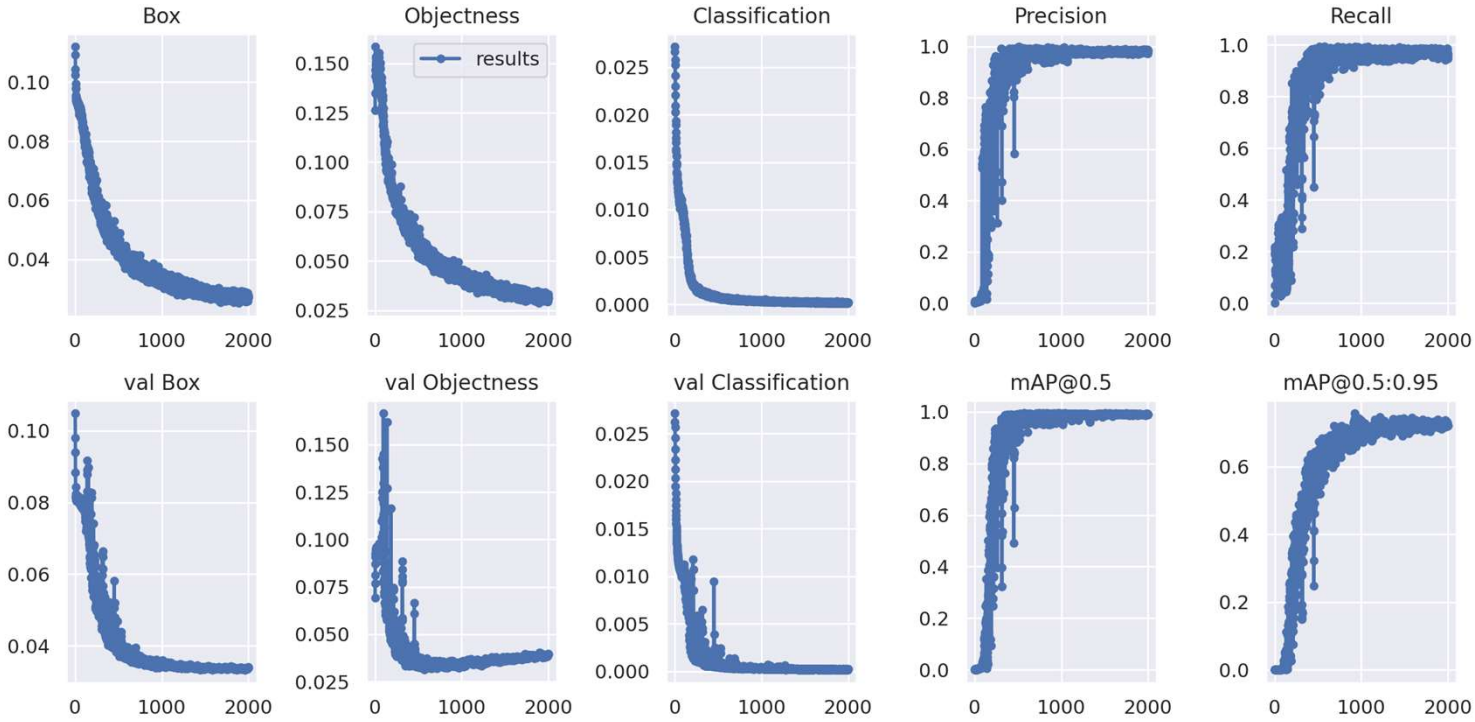




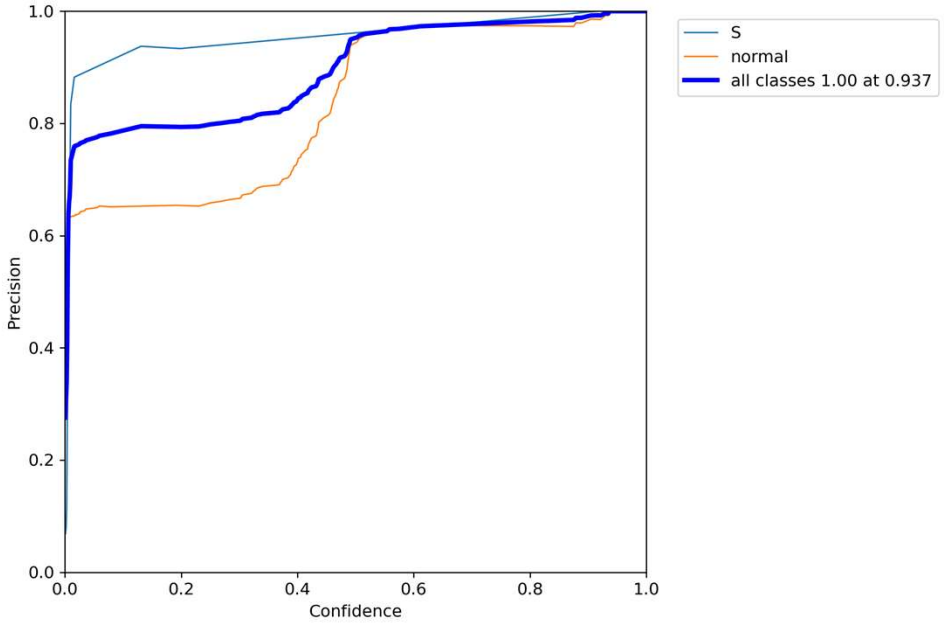
脊椎位置偵測 – 模型訓練

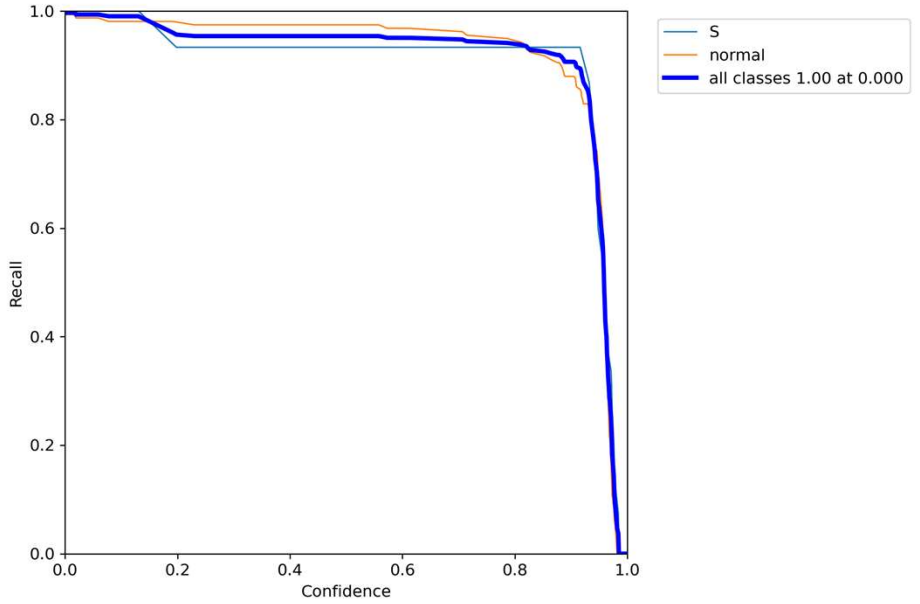
- 模型訓練
- 模型: yolov5m(https://github.com/ultralytics/yolov5)
- 平台: Google Colab (提供免費GPU)
- 資料:
  - Dataset: 168 images and 168 label files
  - Classes: [‘S’(薦椎), ‘normal’]

脊椎位置偵測 – Train Log



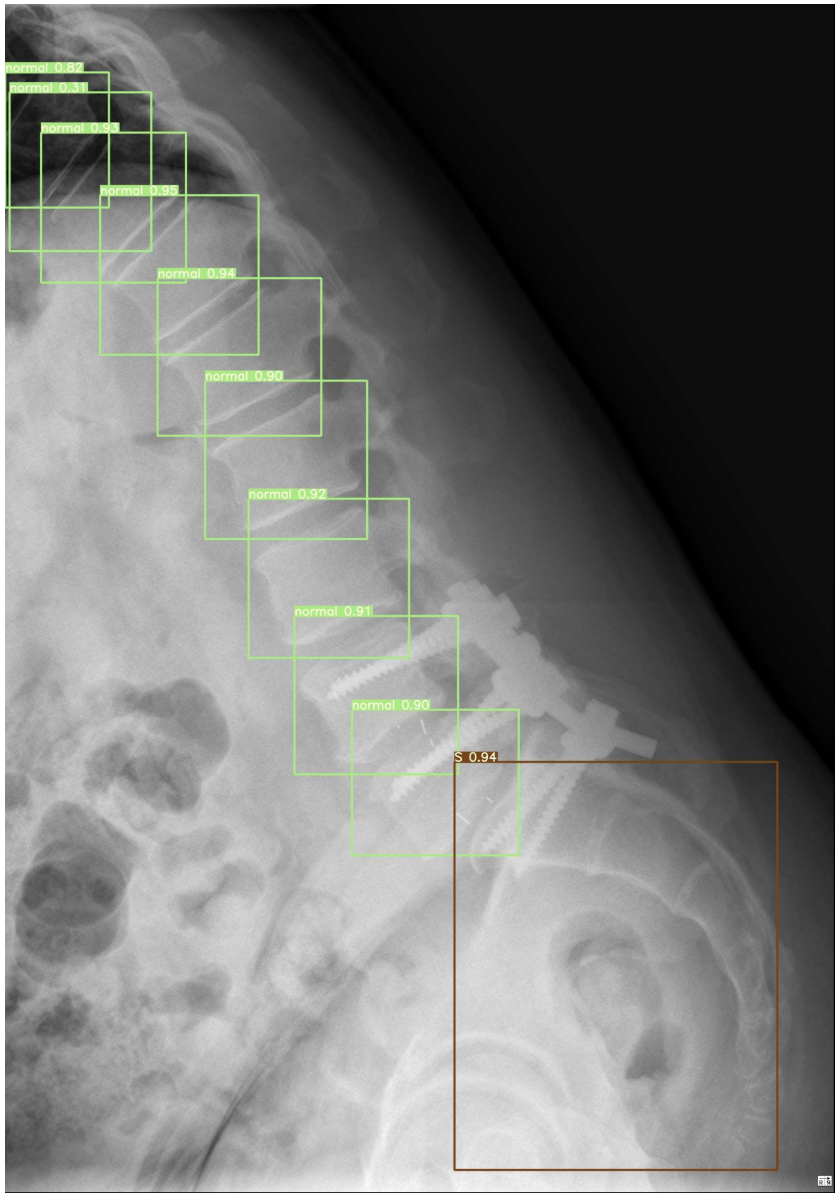
脊椎位置偵測 – Train Log





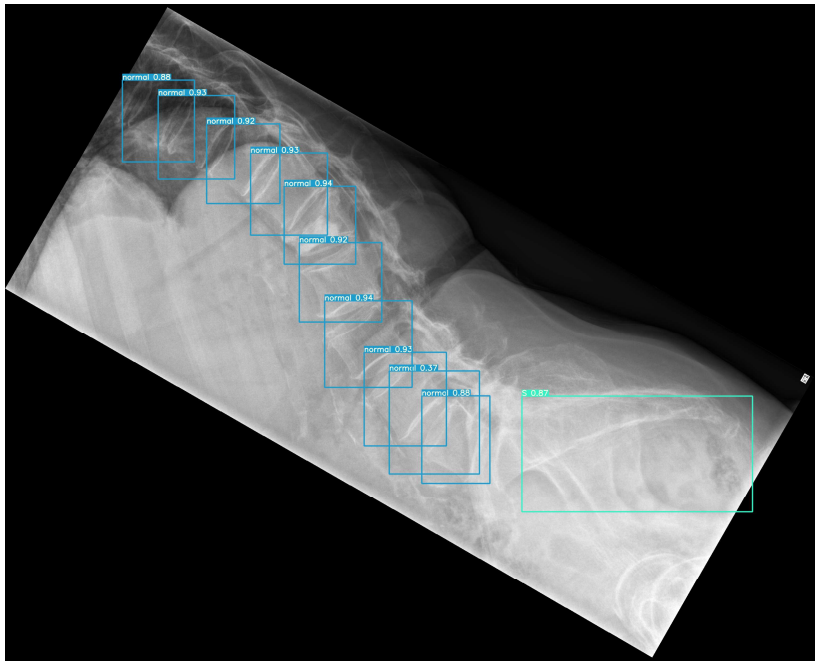
脊椎位置偵測 - 測試結果

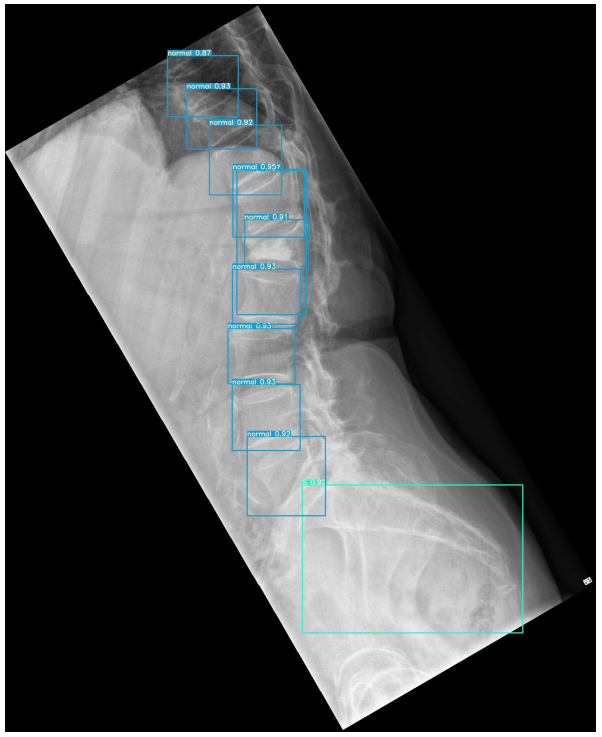
- 效果佳，大致都可以找出所有脊椎，有打鋼釘的脊椎與薦椎也都大致可以辨識，且信心分數都相當高
- Yolov5是設計可以針對影片做影像辨識，因此偵測時的辨識速度快，一張圖不需要一秒
- 仍然有些不準確的部分大多可以透過調高threshold來過濾

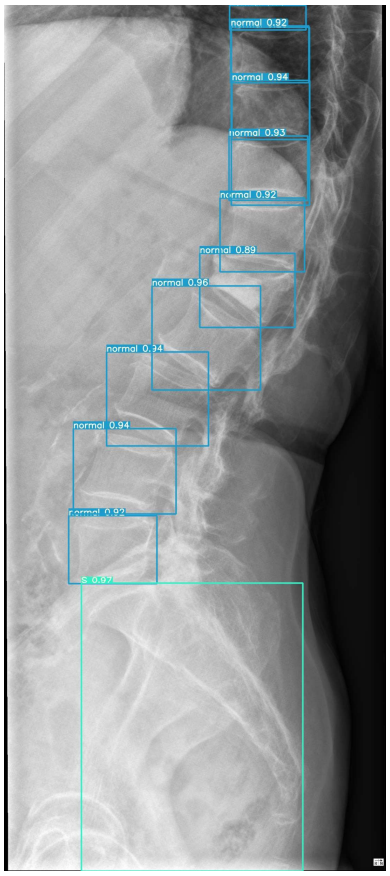


脊椎位置偵測 - 測試結果

- 旋轉資料也可以得到不錯的結果







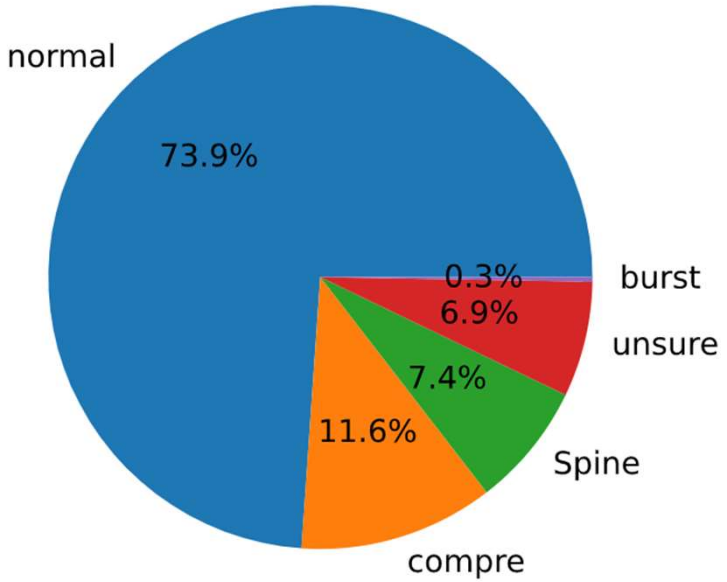
脊椎鋼釘偵測

- 目標: 給定以yolov5模型偵測出的脊椎位置(給定中心點、box的長寬)，crop該截的圖片以模型判斷該截是否有鋼釘。

- 作業內容:
  1. 前處理
  2. 模型訓練
  3. 測試結果

脊椎鋼釘偵測-前處理

- 脊椎Dataset(截)資料分佈
- Normal占了大部分的資料
- 標有鋼釘的資料(unsure)僅占其中的6.9%(102個)，不平衡的資料會導致在分辨上的困難

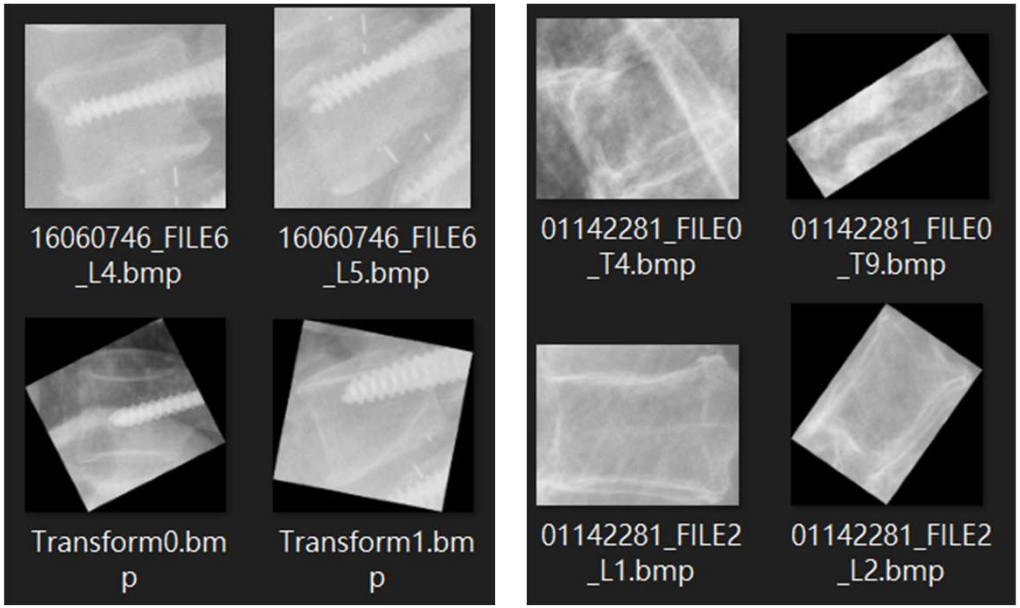


脊椎鋼釘偵測-前處理

- 將鋼釘的Dataset以旋轉的方式做Data expansion，原本102張鋼釘將其旋轉擴增至500張，其他的圖片則隨機選取500張並隨機選轉398張。平衡的資料量通常可以使分類器表現得更好。

- 鋼釘(左)

- 無鋼釘(右)



脊椎鋼釘偵測-模型訓練

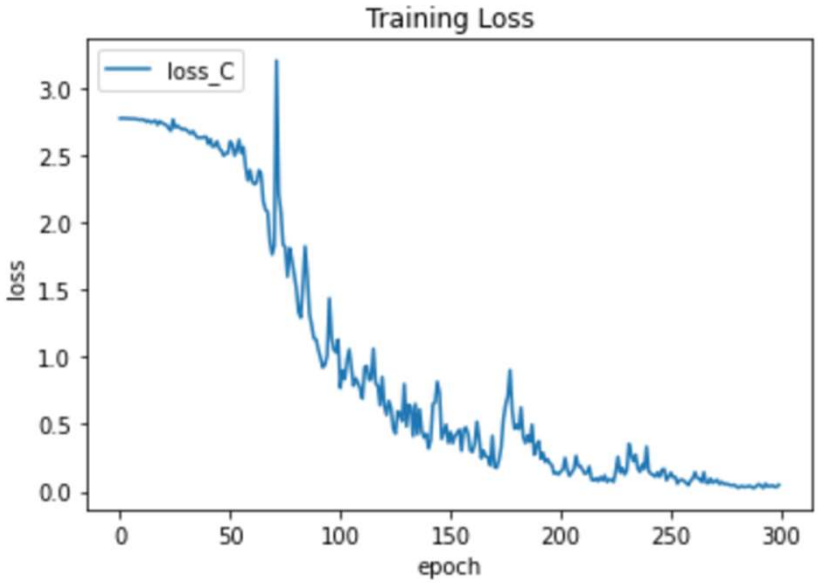
- 使用Alexnet架構，分兩類 [有鋼釘, 沒鋼釘]

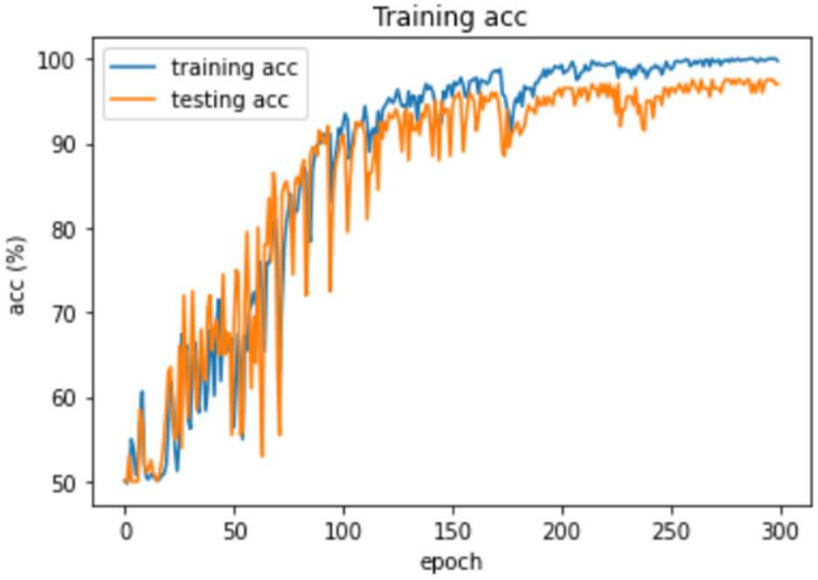
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 55, 55]	7,808
ReLU-2	[-1, 64, 55, 55]	0
MaxPool2d-3	[-1, 64, 27, 27]	0
Conv2d-4	[-1, 192, 27, 27]	307,392
ReLU-5	[-1, 192, 27, 27]	0
MaxPool2d-6	[-1, 192, 13, 13]	0
Conv2d-7	[-1, 384, 13, 13]	663,936
ReLU-8	[-1, 384, 13, 13]	0
Conv2d-9	[-1, 256, 13, 13]	884,992
ReLU-10	[-1, 256, 13, 13]	0
MaxPool2d-11	[-1, 256, 6, 6]	0
AdaptiveAvgPool2d-12	[-1, 256, 6, 6]	0
Dropout-13	[-1, 9216]	0
Linear-14	[-1, 4096]	37,752,832
ReLU-15	[-1, 4096]	0
Dropout-16	[-1, 4096]	0
Linear-17	[-1, 4096]	16,781,312
ReLU-18	[-1, 4096]	0
Linear-19	[-1, 2]	8,194

Total params: 56,406,466
Trainable params: 56,406,466

脊椎鋼釘偵測-Train log

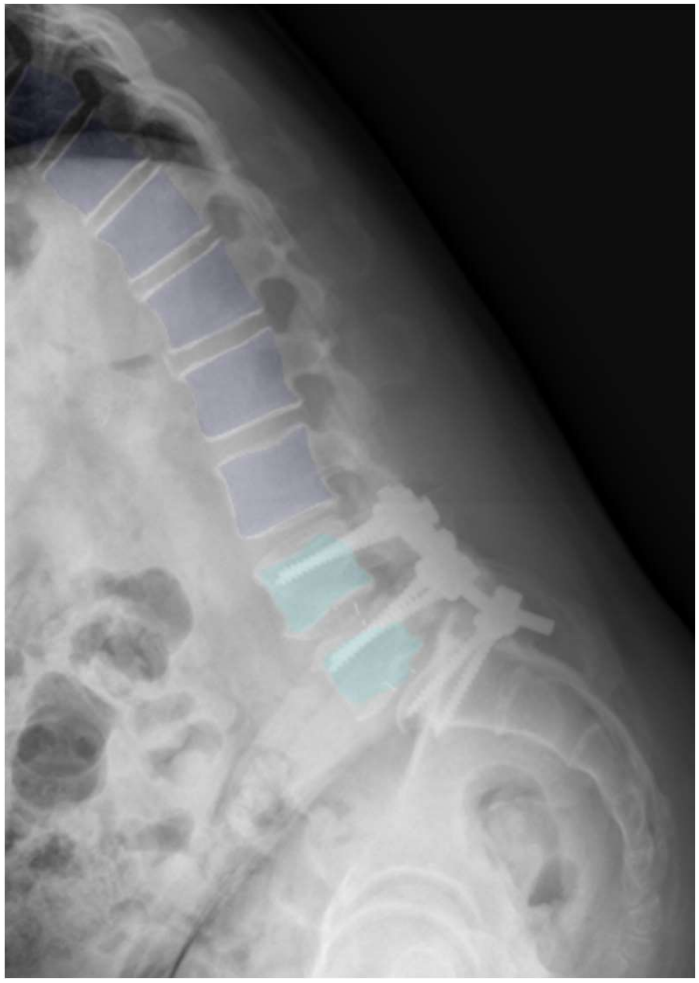
- Train accuracy: 100%
- Test accuracy: 97%
- Accuracy start with 50% because it’s a binary classifier





脊椎鋼釘偵測-測試結果

- 深藍色: 無鋼釘
- 水藍色: 有鋼釘
- 薦椎部分因不在討論範圍不進行偵測



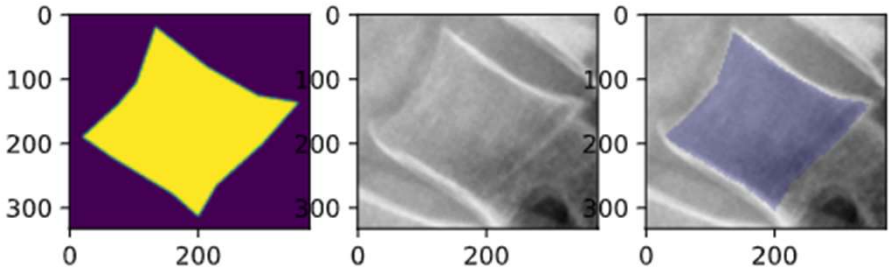
脊椎邊界偵測

- 目標: 偵測脊椎的形狀，藉由脊椎的形狀萃取出有用的資訊，如脊椎(單截)的左邊長、右邊長、中間長度等等可以做為判斷是否骨折的feature。

- 作業內容:
  1. 前處理
  2. 模型訓練
  3. 後處理

脊椎邊界偵測-前處理

- 訓練資料: 將每一截脊椎xy的bounding box在稍微加寬一些並截取出來成一張圖片
- 訓練標記: 在資料標記時每一節脊椎以12個點(四個角+每個邊兩點)組成的邊界所圍成的圖形
- Mask(左)、原圖(中)、Mask疊圖(右)
- 圖為00048035_FILE0_L2.bmp

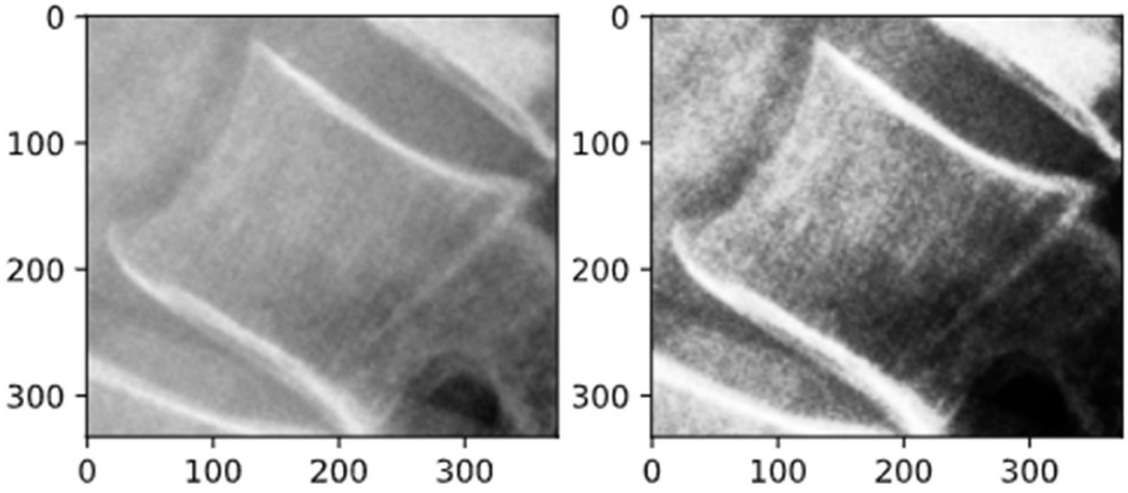


脊椎邊界偵測-前處理

- Histogram Equalization，由於脊椎的邊界通常會是一張圖片特別亮的部分，透過Histogram Equalization的技術可以加強對比度，使得脊椎形狀的判斷較為容易

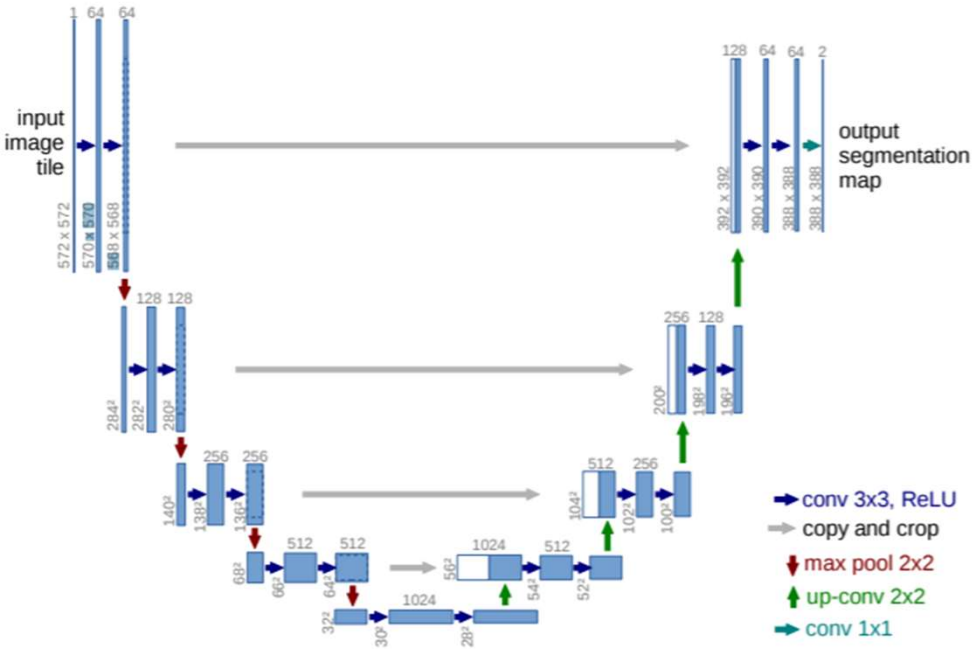
- 原圖(左)

- Histogram Equalization(右)



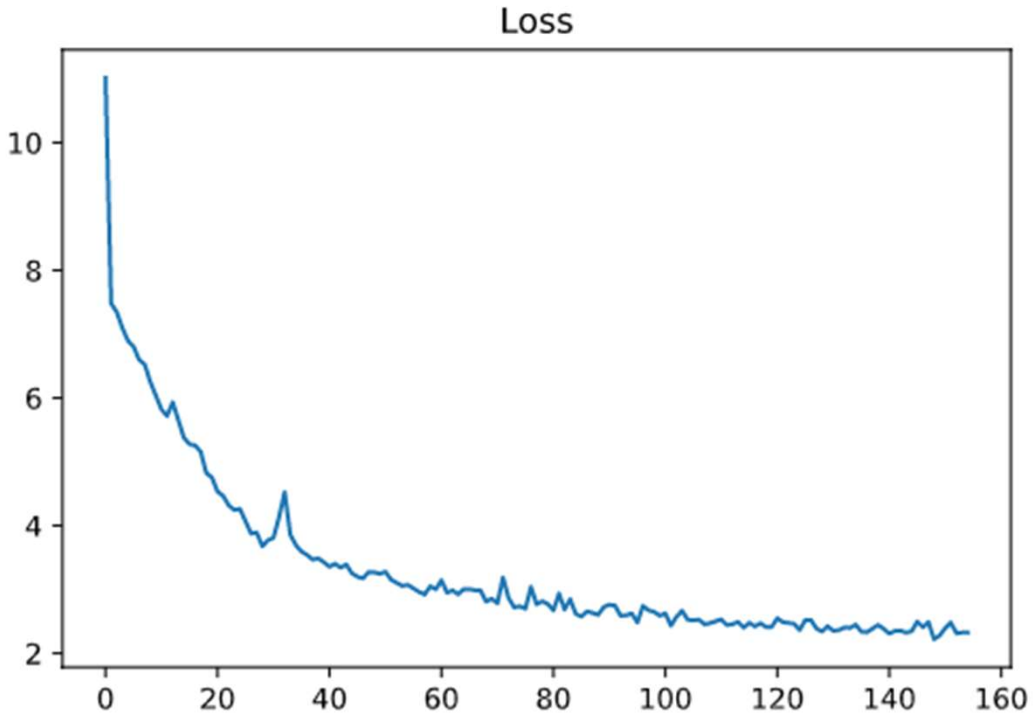
脊椎邊界偵測-模型訓練

- 目前採用Unet
- 不分脊椎類型
- 圖片來源



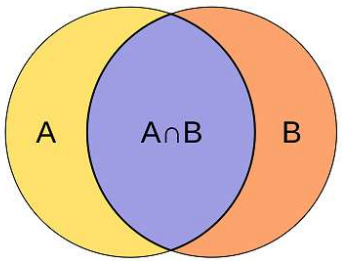
脊椎邊界偵測-Train log

- Loss-epoch
- 使用BCELoss

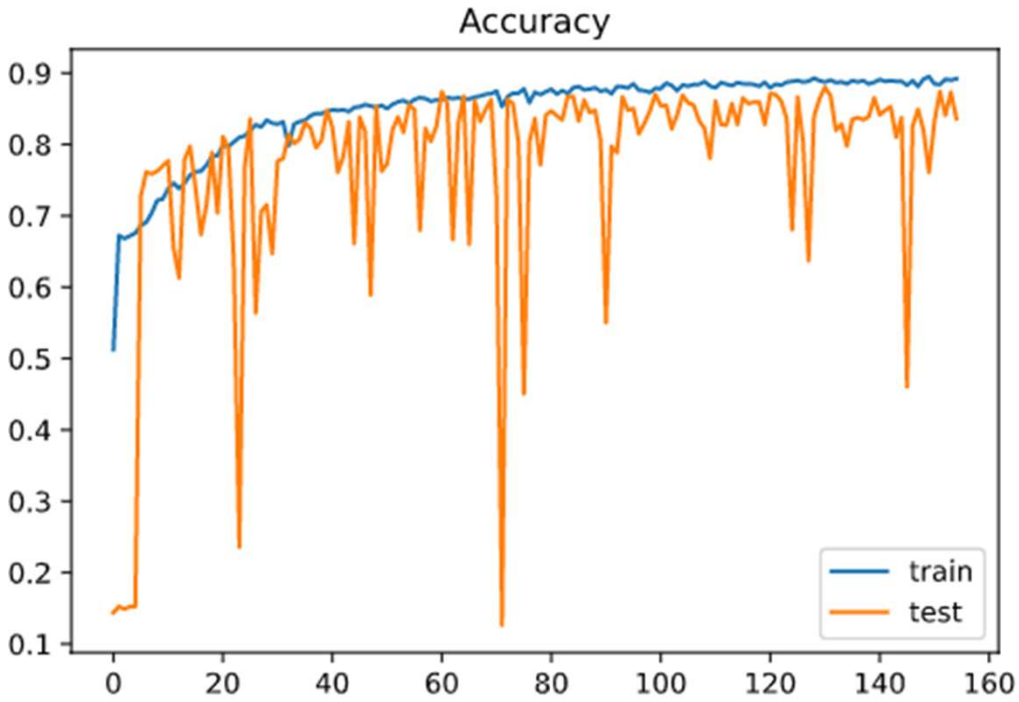


脊椎邊界偵測-Train log

- Accuracy-epoch
- 計算方式: Dice coefficient
- $Dice\ coeff = \frac{2TP}{2TP+FP+FN}$

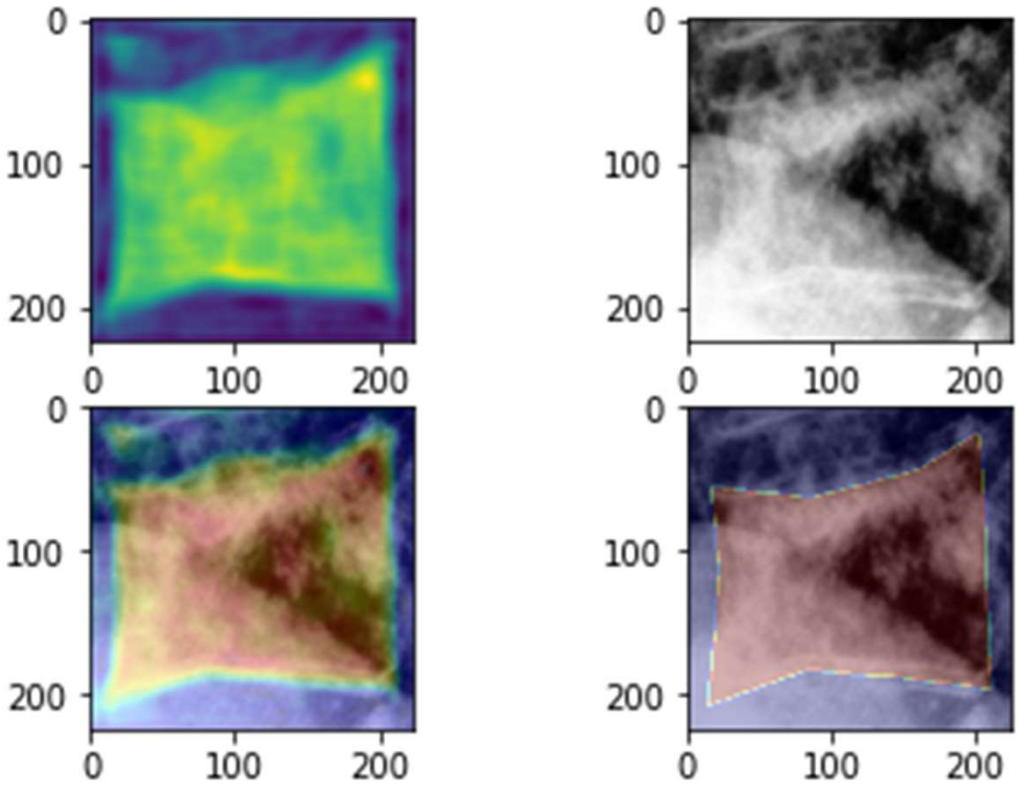


$$\text{Dice coefficient}(A,B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$



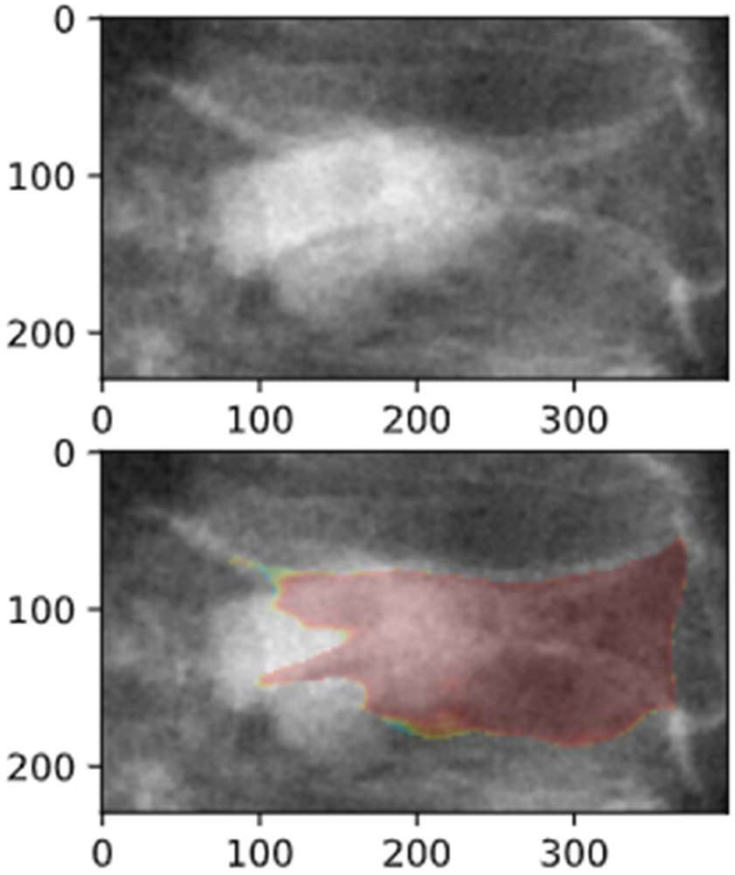
脊椎邊界偵測-測試結果

- 大致上效果不錯，所輸出的影像(左上)大致與原本標記的mask(右下)相等
- 完成一張圖的偵測時間需要數秒
- 對於輪廓清楚的正常脊椎判斷效果尤佳，但對於輪廓模糊的脊椎則仍有誤判情況(下頁)



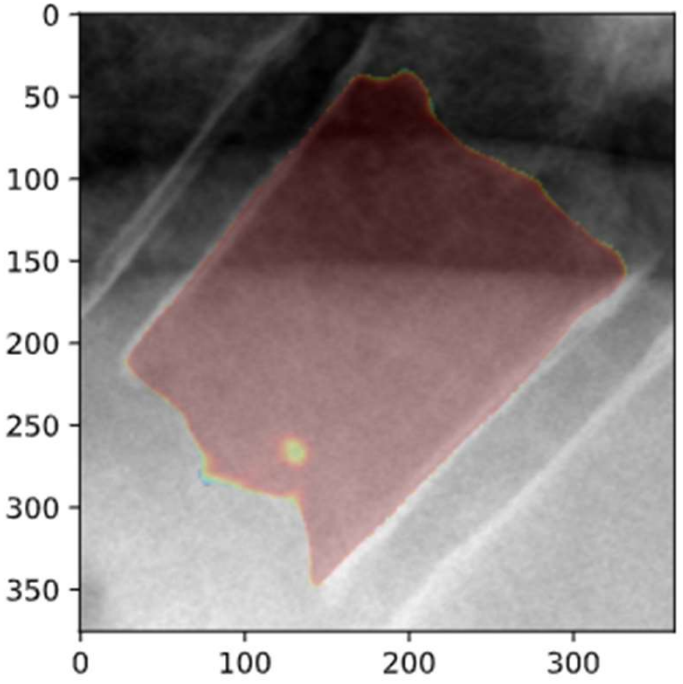
脊椎邊界偵測-測試結果

- 邊界不明顯的骨折脊椎
- 右側的右上角跟右下角仍然有不錯，甚至是良好的判斷結果
- 中間大量白色的部分被視為是脊椎的邊界使得Mask無法覆蓋到左上角的部分可能導致錯誤
- 中間的下方邊界應該更窄



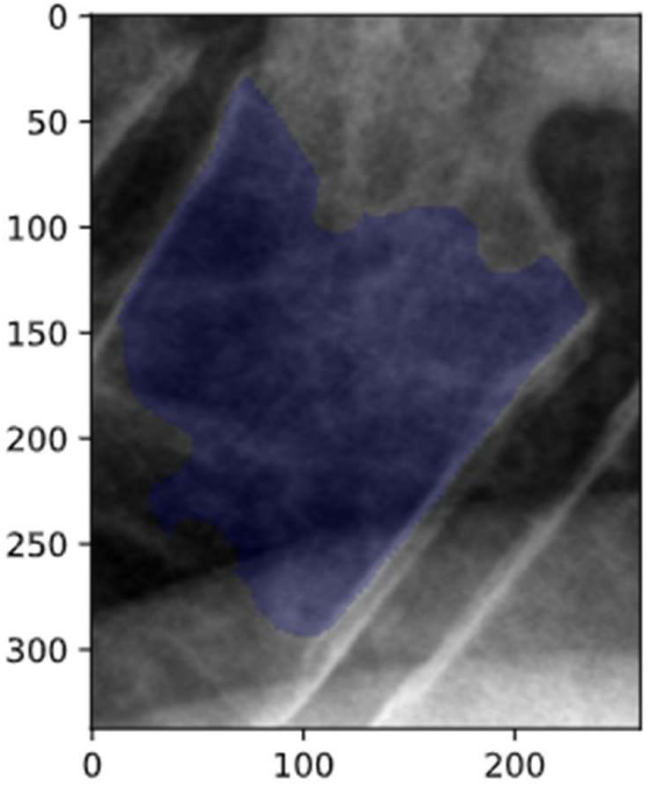
脊椎邊界偵測-測試結果

- 儘管大多數時候可以精準的判斷脊椎外型，但有時仍然會有有些小瑕疵
- Unet輸出結果為根據每個pixel的分數是否超過threshold判斷，因此未必會是完整圖形
- 可能有多餘的凸角或是空缺的部分



脊椎邊界偵測-測試結果

- 不完整的圖片容易發生錯誤
- 通常發生在圖片的第一節脊椎
- 沒有拍攝完整的脊椎通常不會是關注的重點，可以忽略



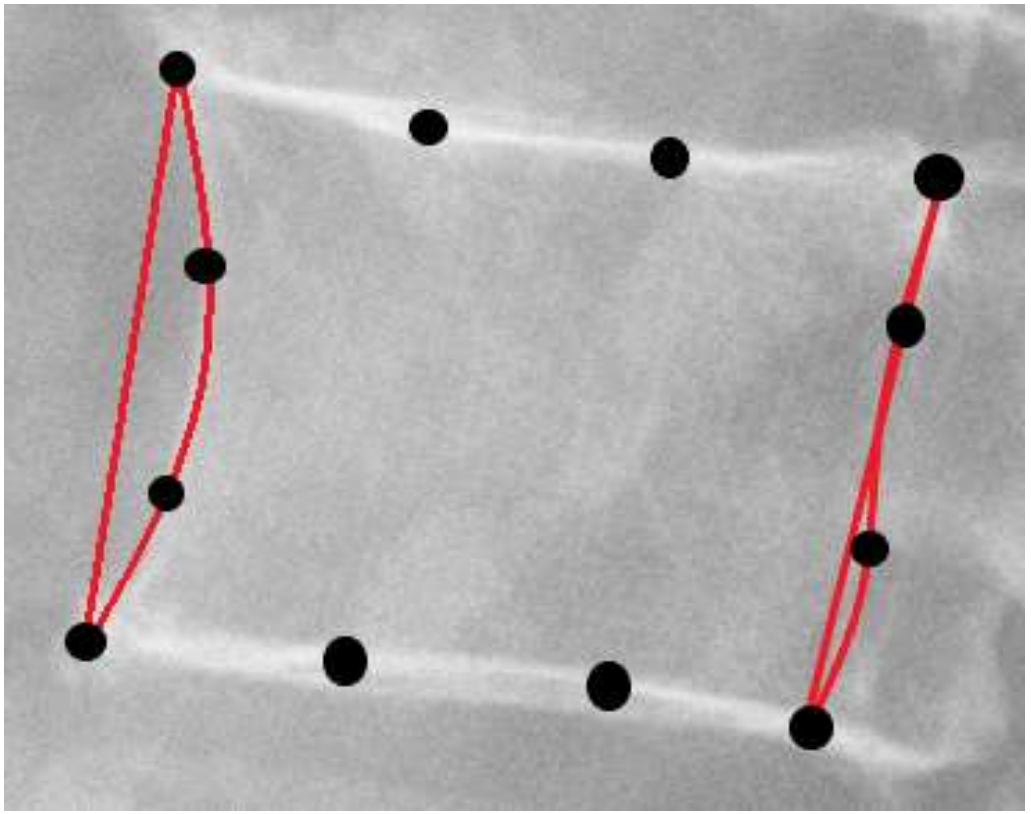
脊椎狀況判斷

- 目標: 藉由前面所有的偵測模型可以從一張完整的X光影像中萃取出有用的資訊，並且使用資訊以機器學習/深度學習的方式分析圖片 進行脊椎裝況的判斷

- 作業:
  1. 標記特徵處理
  2. 資料分析
  3. 模型訓練
  4. 未標記特徵萃取
  5. 測試結果

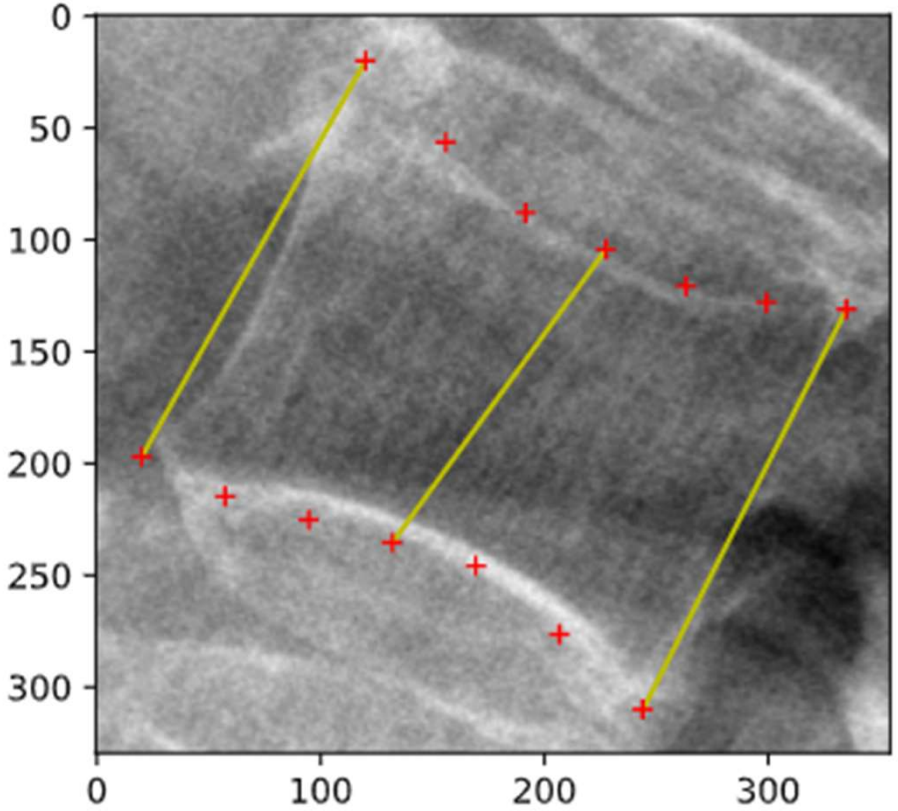
脊椎判斷-標記特徵處理

- 兩邊長度
- 標記邊長由12個點的x,y座標組成(如圖)
- 兩種選取方式
  - 兩個角的距離
  - 左邊四個點之間直線距離和(目前主要採用)



脊椎判斷-標記特徵處理

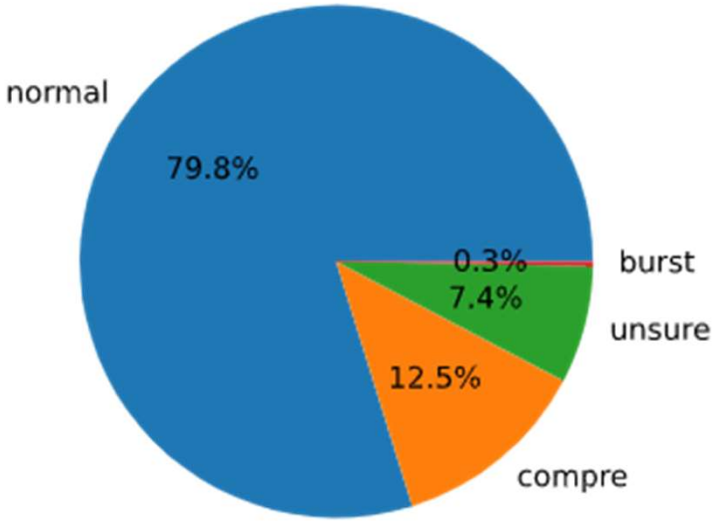
- 目前將單截脊椎分六段
- 取每一個上下對應點的長度作為特徵
- 取中間/左邊、右邊/中間、右邊/左邊
- T(胸椎)=0，L(腰椎)=1



脊椎狀況判斷-資料分析

- 先分析原本手動標記的資料作為模型訓練的基礎

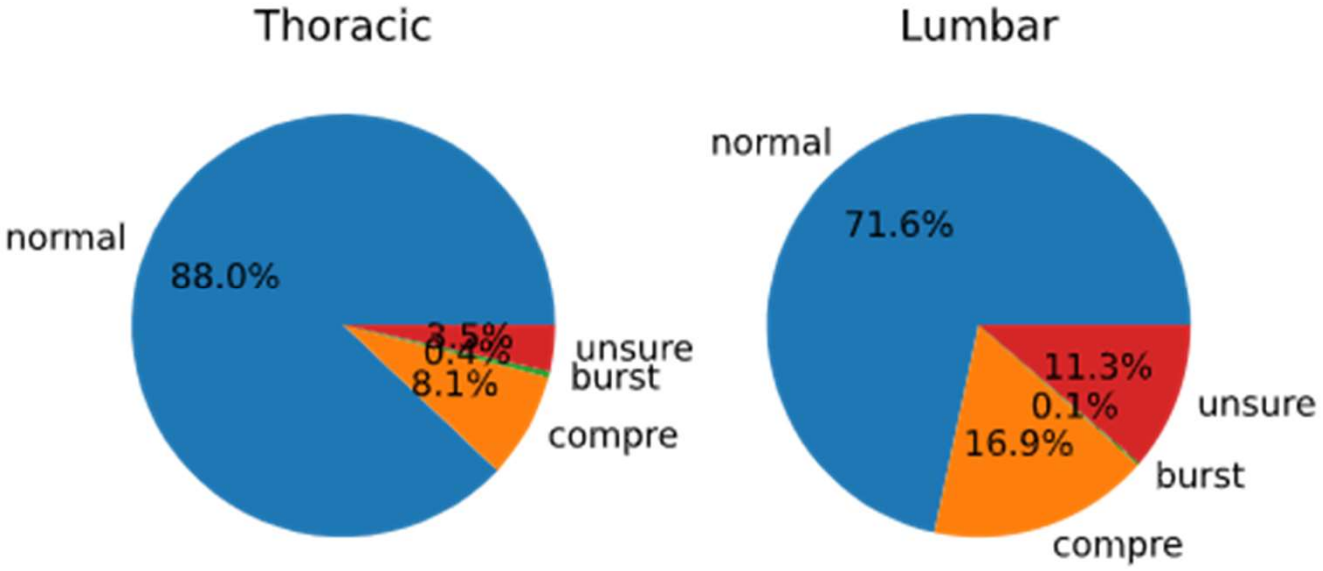
- 總共四種類別:
  1. Noramal: 正常 (1098)
  2. Compre: 壓迫性骨折 (172)
  3. Burst: 爆裂性骨折 (4)
  4. Unsure: 打鋼釘或是不確定 (110)



脊椎狀況判斷-資料分析

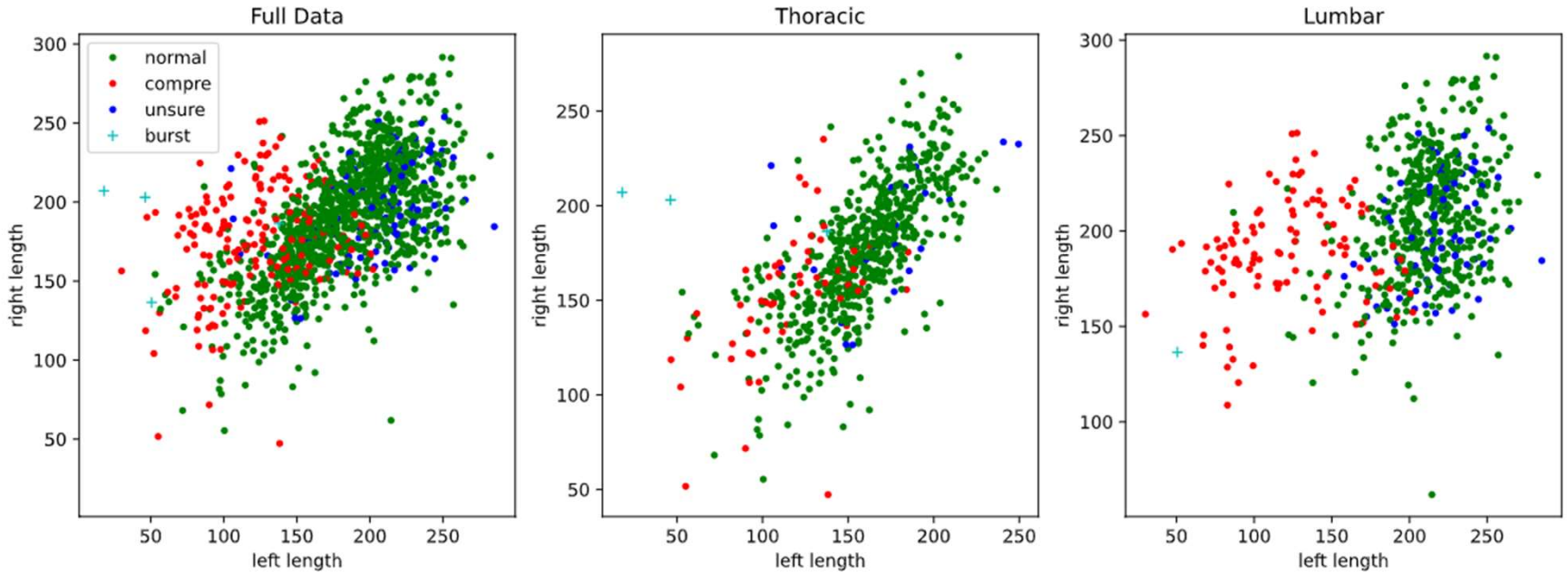
- 將資料分成 胸椎(Thoracic) 和 腰椎(Lumbar)

- 腰椎的資料較為豐富，較利於做學習預測



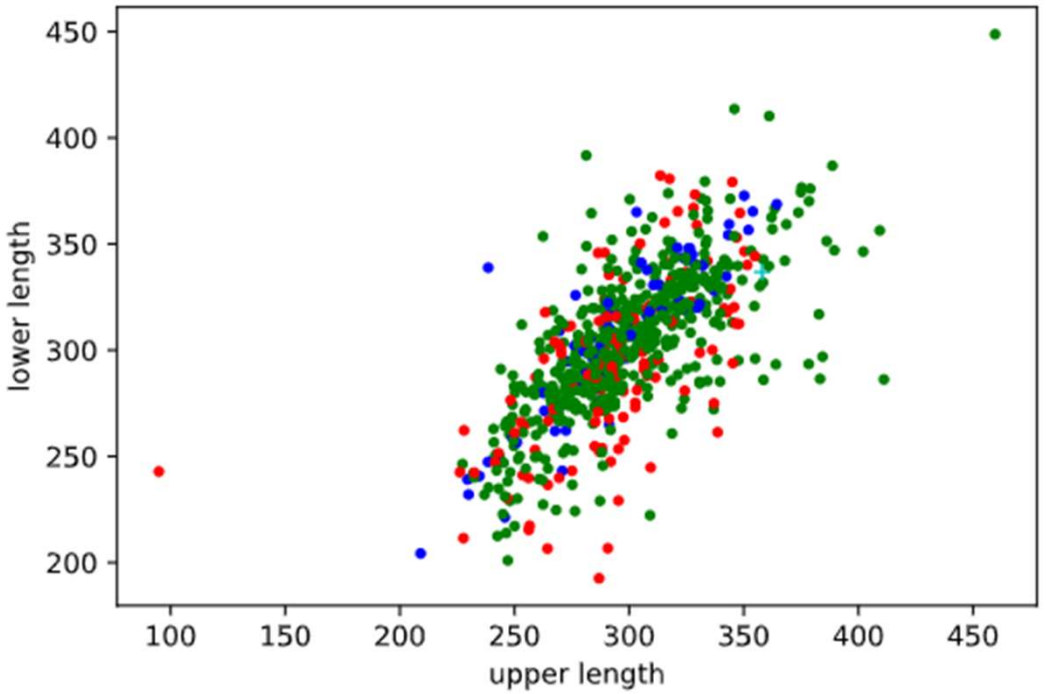
脊椎狀況判斷-資料分析

- 從左右邊長度的散佈圖看資料，胸椎的資料點較難以判斷，但約略可以看見紅點都在較下半，腰椎則有較顯著差別



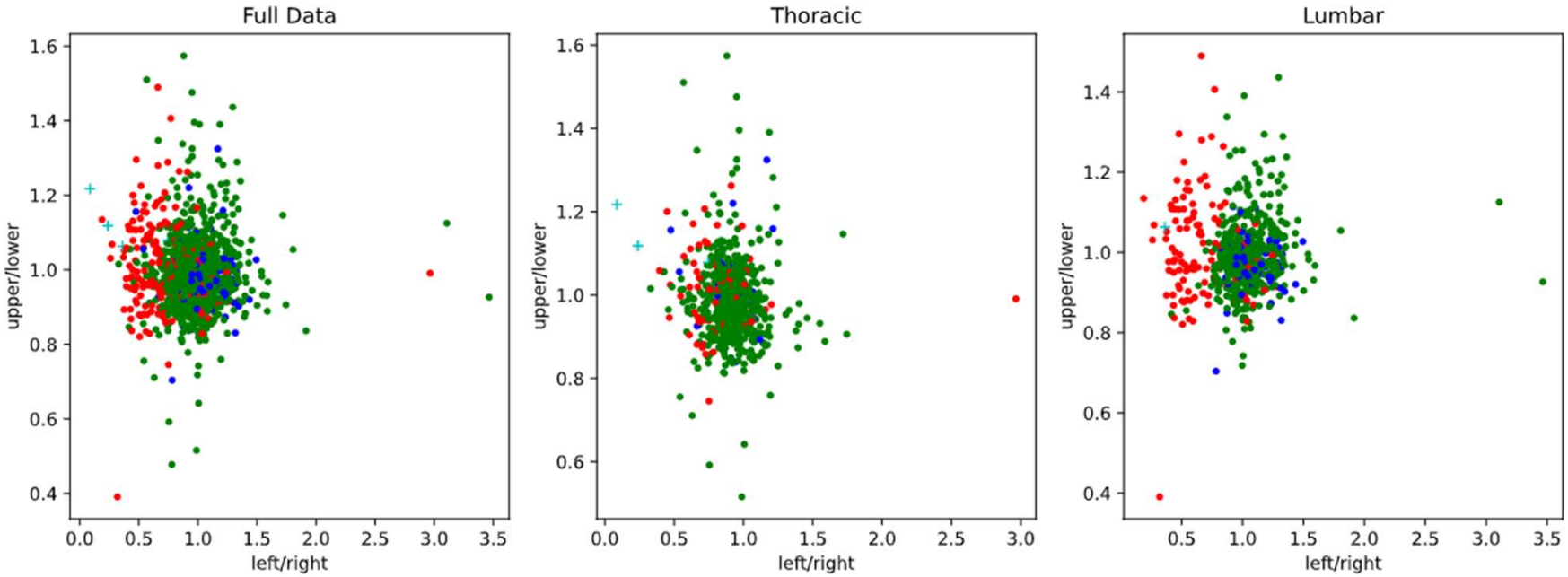
脊椎狀況判斷-資料分析

- 從上下長度的散佈圖則有嚴重混淆的現象



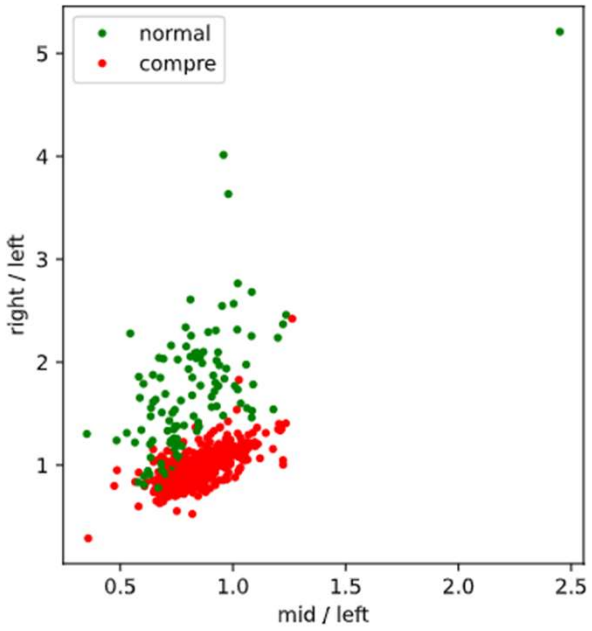
脊椎狀況判斷-資料分析

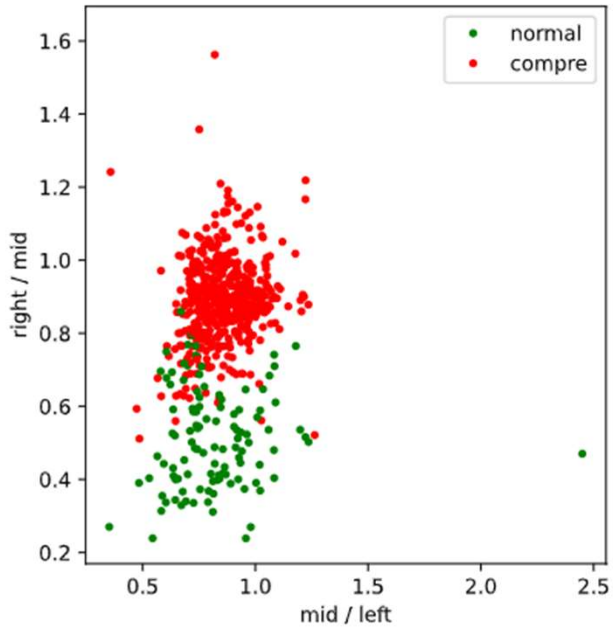
- X軸左右比值 Y軸上下比值，在腰椎有顯著差異，但在胸椎也同樣差異不顯著



脊椎狀況判斷-資料分析

- 醫生建議脊椎高度的Ratio是很好的特徵
- Mid/left對分布相關性似乎不大
- Right/left和right/mid則看起來與骨折與否相關性強



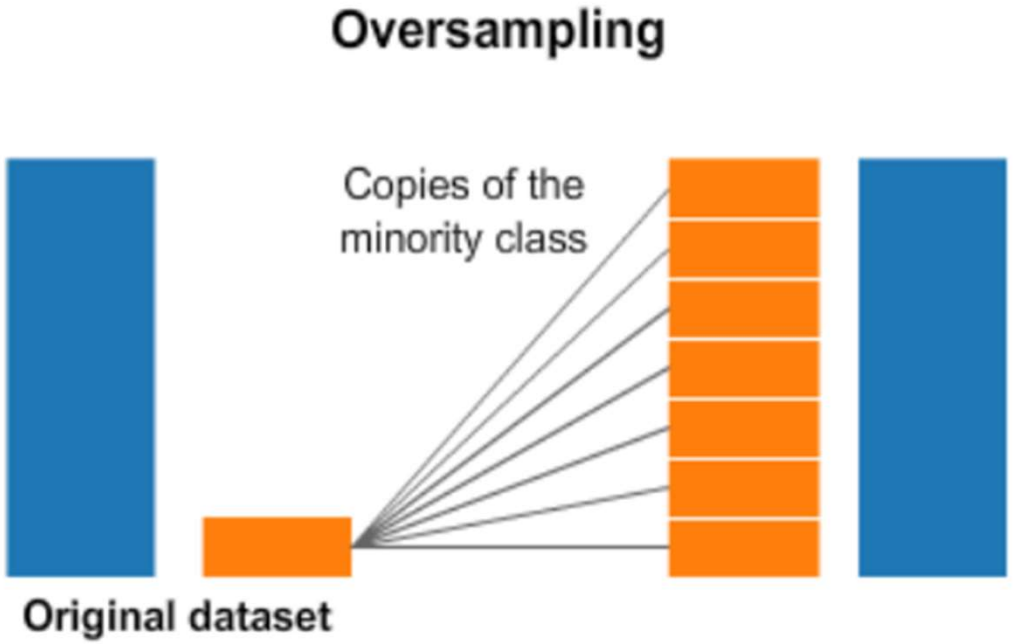


脊椎判斷-模型訓練

- 資料: 所有手動標記之資料(根據醫生的病例判斷)
- 特徵: 長度、長度比值、腰椎或胸椎(如前頁所述)
- 模型: KNN, DecisionTree, RandomForest
- 方法:
  - 80% train、20%test
  - 使用Oversampling與cross validation (7 folds) 方法
- 其他: 骨折部分不分壓迫另與爆裂性(爆裂性骨折資料過少)

脊椎判斷-模型訓練

- Oversampling
- 將嚴重缺乏的資料(骨折重複複製
- 之後做cross validation的時候使用每個fold都有兩種資料個50%



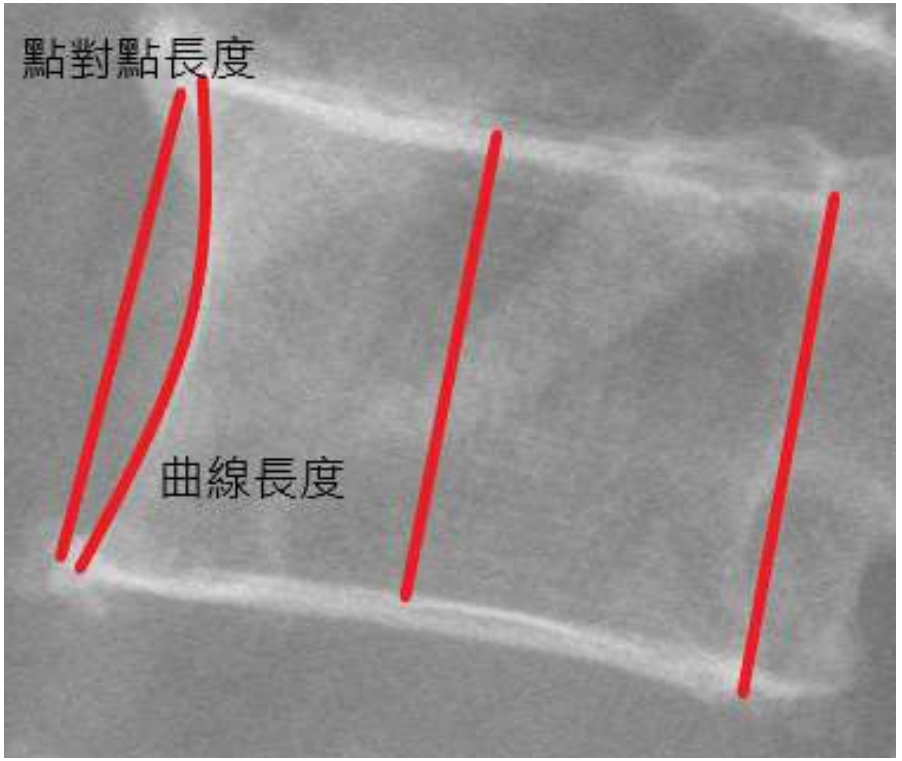
脊椎判斷-模型訓練

- 訓練結果: (accuracy)
- 上方為每個fold在test set上的結果
- 下方為7個fold合起來的結果與單純holdout的模型

```
knn     dec     rfc
0.948   0.952   0.964   | # of train data 1536
0.948   0.96    0.96    | # of train data 1536
0.944   0.964   0.96    | # of train data 1536
0.944   0.956   0.964   | # of train data 1536
0.948   0.976   0.956   | # of train data 1536
0.952   0.968   0.964   | # of train data 1536
0.948   0.964   0.956   | # of train data 1584
--------------7 fold validation----------------
0.948   0.972   0.964   | 7 fold on 1800 data
-----------------full input--------------------
0.948   0.96    0.96    | train on 1800 data
```

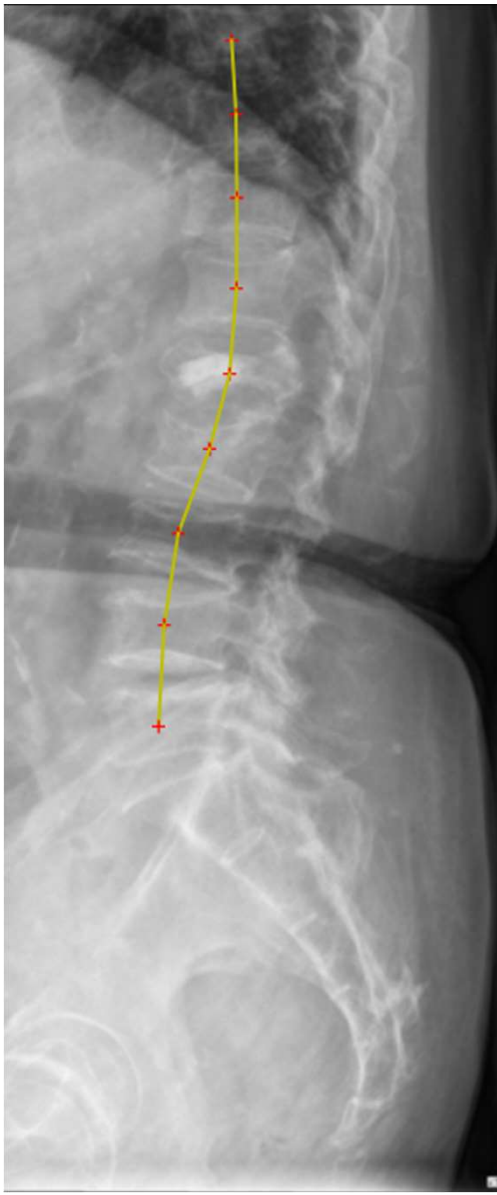
脊椎狀況判斷-未標記特徵萃取

- 目前已知重要特徵
- 兩邊的長度
  1. 角落對角落直線長度
  2. 連續曲線長度
- 中間的長度
- 其他特徵如上下邊長、對角線長、脊椎位置等等



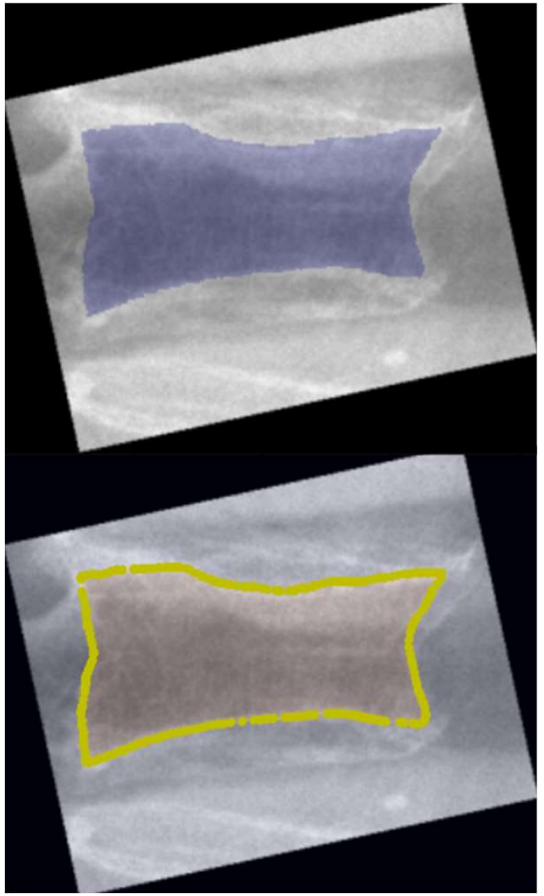
脊椎狀況判斷-未標記特徵萃取

- 標記每一節脊椎的中心點
- 計算每一截脊椎之間的斜率
- 若上下皆有脊椎則將兩個斜率平均，否則取有存在那個斜率
- 記錄每一節的斜率與計算角度旋轉使得正斜率為Y軸



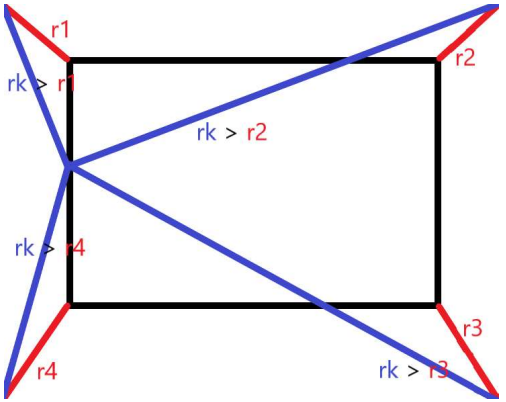
脊椎狀況判斷-未標記特徵萃取

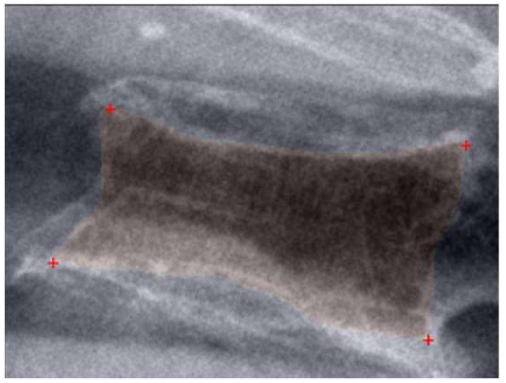
- 將每一截脊椎切割成單獨圖片
- 取L4(倒數第2截)為例
- 使用先前的角度將單截圖片 做角度校正
- 使用cv2.findcontour找尋圖形的contour(若因為形狀不完整而找到多個contour則取節點數最多的那個contour)



脊椎狀況判斷-未標記特徵萃取

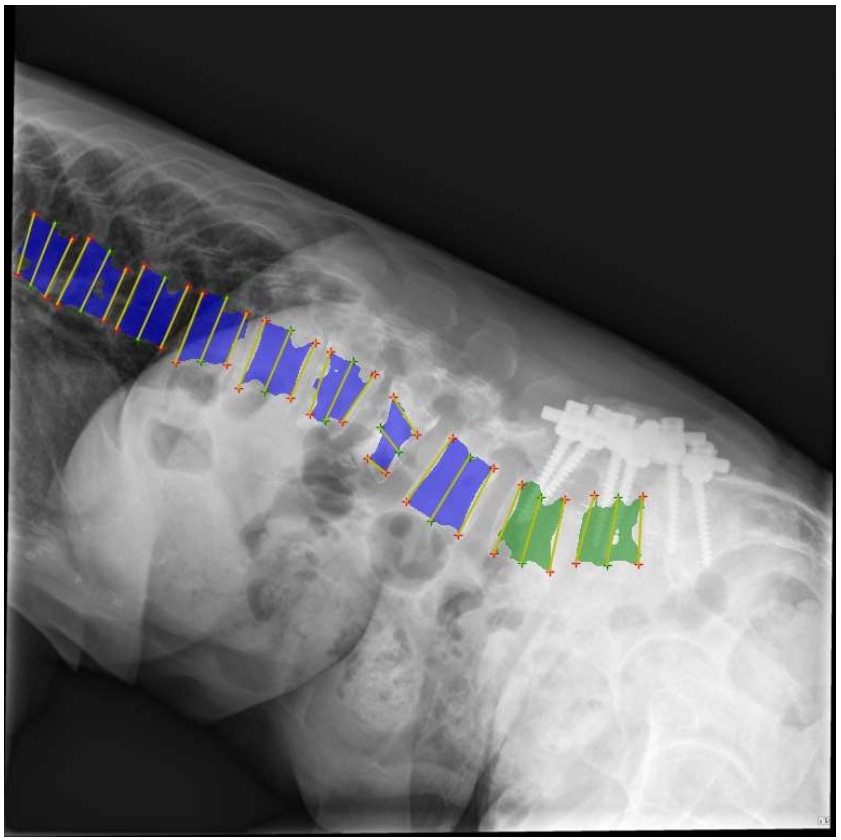
- 目標要取得四個角
- 使用MinCornerDistance，及分別與圖片的四個角落距離最短的四個點會被選擇為該截脊椎的Corner
- 由於圖片已經經過角度校正，理想上來說脊椎會呈現近似於正矩形的樣態(右上圖)，因此可以使用此方法得到四個角
- 結果呈現如右下圖





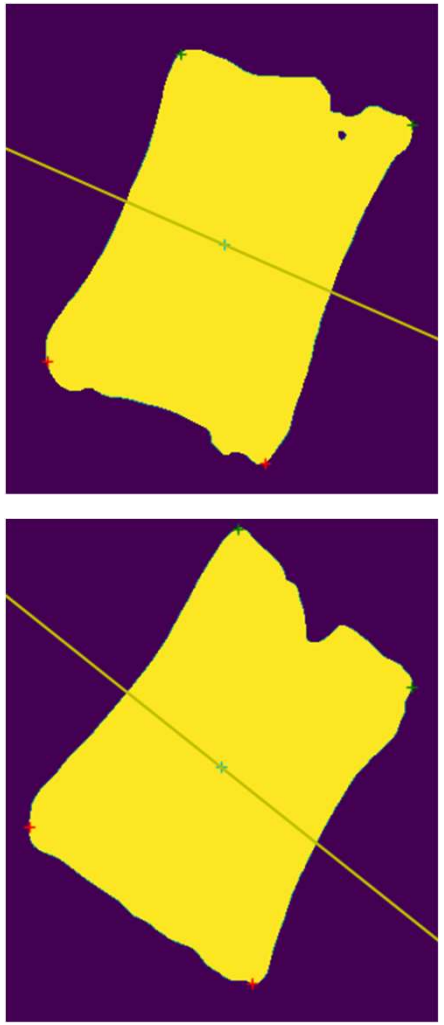
脊椎狀況判斷-未標記特徵萃取

- 特殊情況: 非直立脊椎
- 同樣的斜率與MinCornerDistance的method可以找出四Corner
- 但會誤判每一個corner的順序
- Corner的順序會影響後面部分的feature萃取與判斷(右圖)



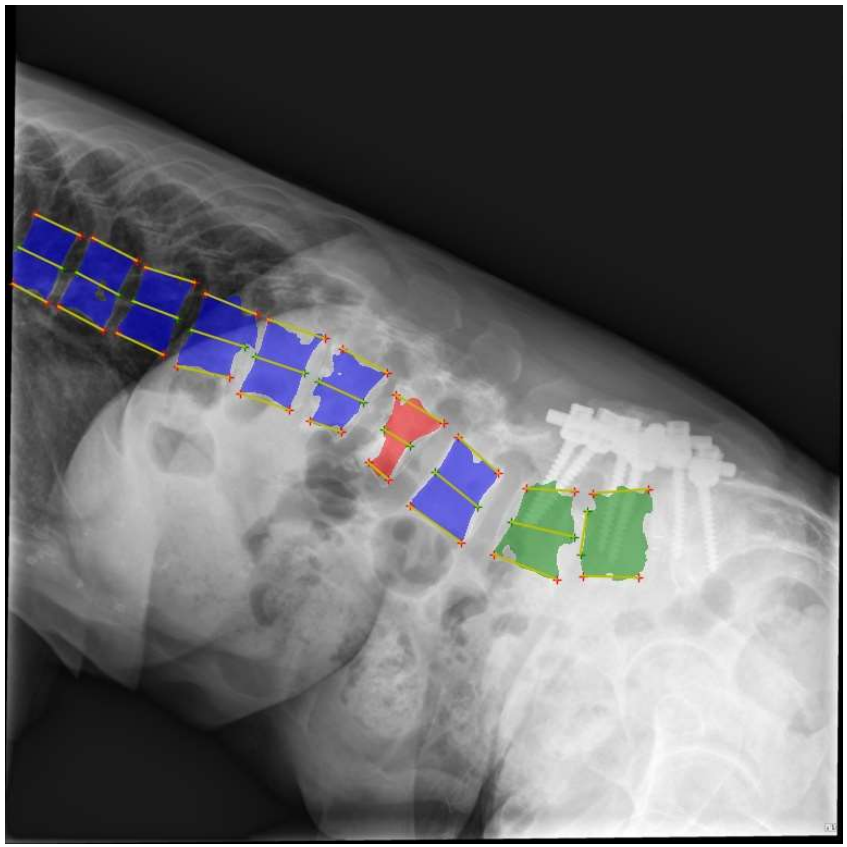
脊椎狀況判斷-未標記特徵萃取

- 在完成MinCornerDistance的時候重新排列四個點
- 根據斜率與圖片與中點畫出一條直線分割左邊與右邊ax+b=y
- 上則脊椎則依照y座標大小即可，不會有混淆情況
- 若$ax_1+b-y_1 < 0$為左邊(紅色)反之為右邊(綠色)



脊椎狀況判斷-未標記特徵萃取

- 根據中線修正順序後的結果可以順利完成後續的步驟
- 不會影響原本的正確的案例

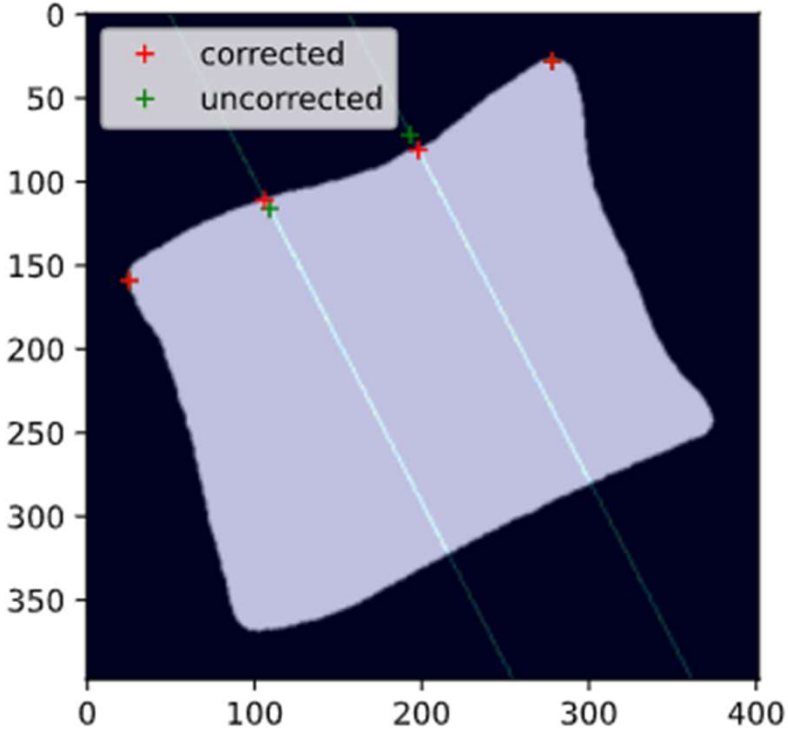


脊椎狀況判斷-未標記特徵萃取

- 中間線的判斷
- 判斷脊椎的形狀時可能需要其中間的高度作為依據
- 儘管有脊椎角落的位置但脊椎角落之間並非直線，直接求中間點未必可以得到正確的位置
- 使用與脊椎角落連線的垂直向量尋找在脊椎上的投影點(下頁範例)

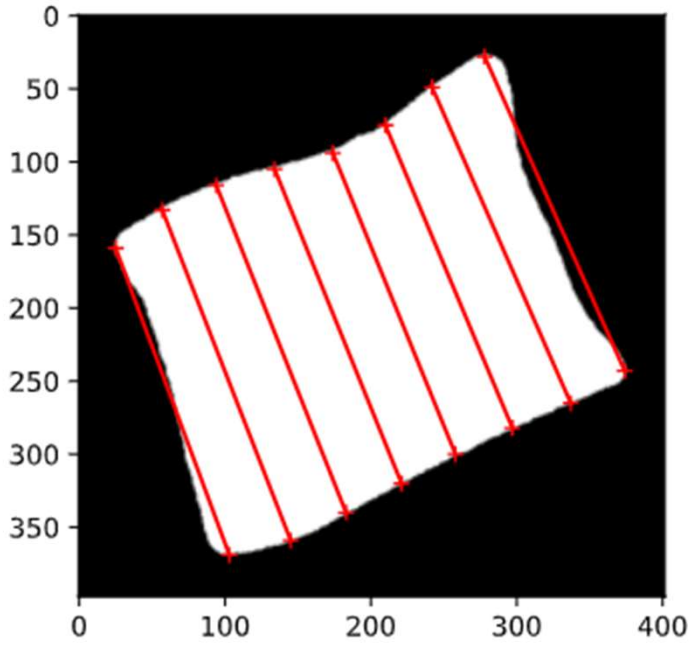
脊椎狀況判斷-未標記特徵萃取

- 以脊椎上緣切成三分為例(n=3):
- 假設左上角(x1,y1), 右上角(x2,y2)
- 向量(x2-x1, y2-y1)的垂直向量(y2-y1, x1-x2)
- 第k個點為sqrt((x1-x2)^2 + (y1-y2)^2)* k/n + c*(y2-y1, x1-x2)
- 第一個碰到脊椎的即決定c的Value與投影的座標值



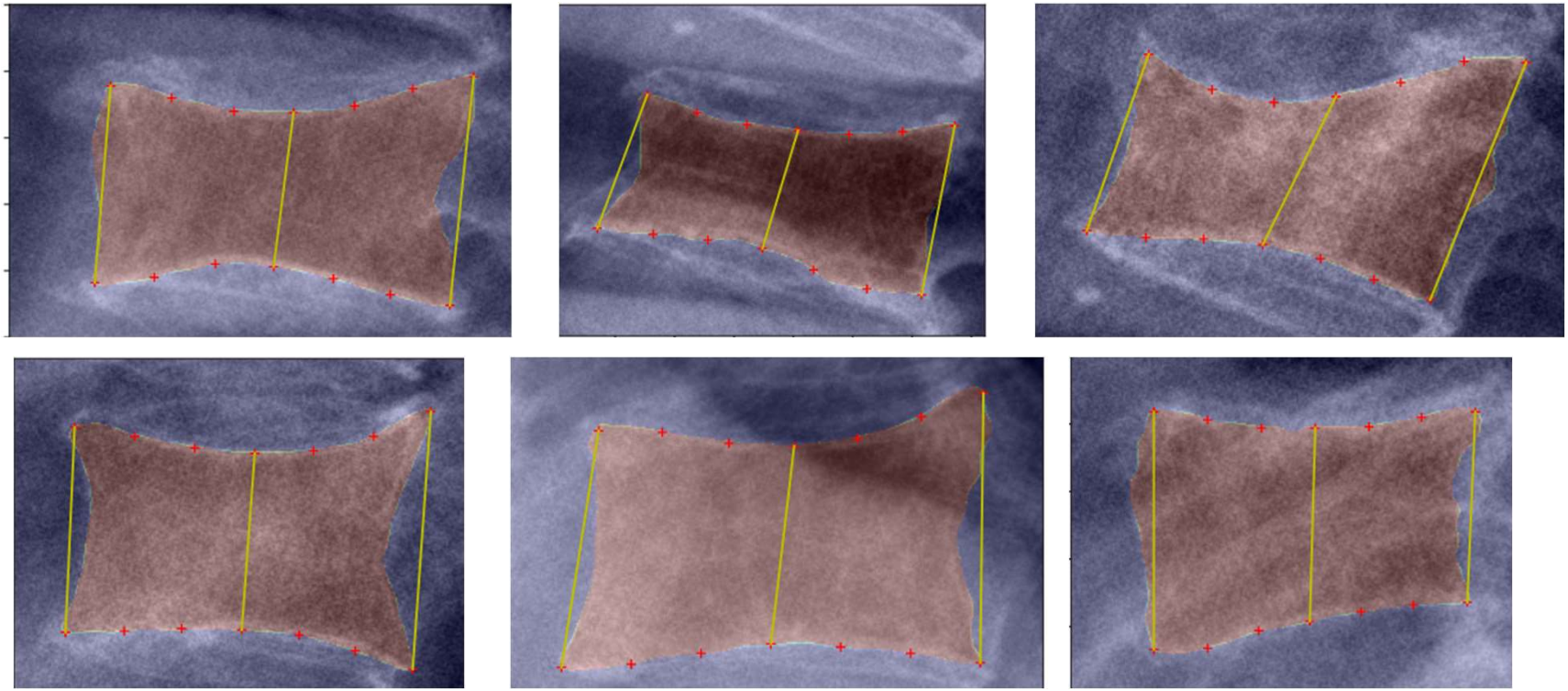
脊椎狀況判斷-未標記特徵萃取

- 延伸至上下緣做座標投影，並且將其作連線(如圖n=7)，即可得到中間線的feature



脊椎狀況判斷-未標記特徵萃取

- 實際應用於測試圖片



視覺化與執行

- 將完整流程包含於一個Vdetect.py檔案

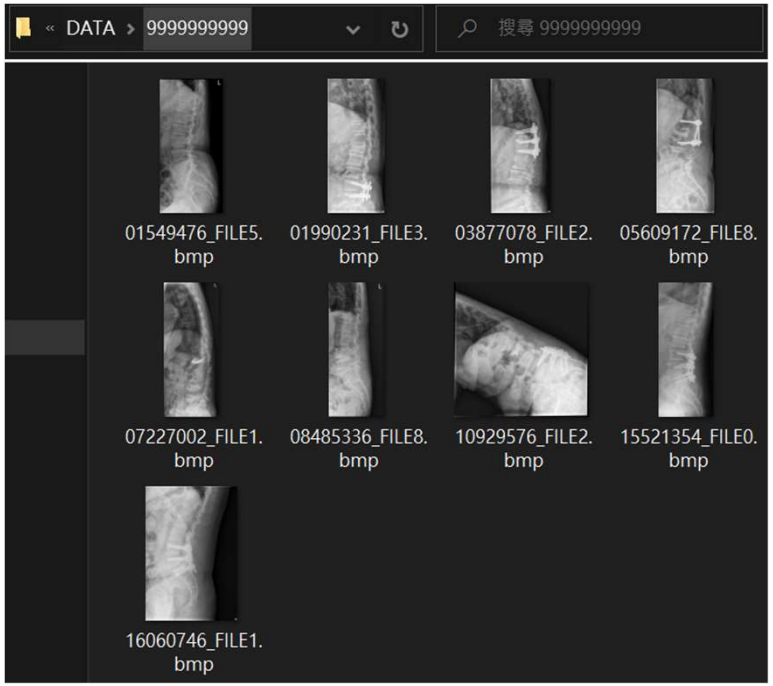
- Vdetect.py
  - --source : 愈偵測的圖片或資料夾路徑
  - --dest: 愈處存的project資料夾路徑
  - --name: 愈儲存於project資料夾內的路徑
  - --mask: 預先儲存好的mask資料夾路徑，預設None
  - --object: 預先儲存好的yolo format file資料夾路徑，預設None
  - --method: 愈使用的方法，knn, rfc, dec, rule，預設rfc

視覺化與執行

- screwDetect.py:
  - get_scew_model : 取得模型
  - screwPredict: 預測
- fractDetect.py:
  - get_fmodel: 取得模型
  - fracturePredict: 判斷骨折
- segfunct.py:
  - edge_segment: 取得兩點連線中間的於mask上的點
  - minCornerDistance: 得到Corner
- visualization.py:
  - ResultVisiualization: 結果視覺化

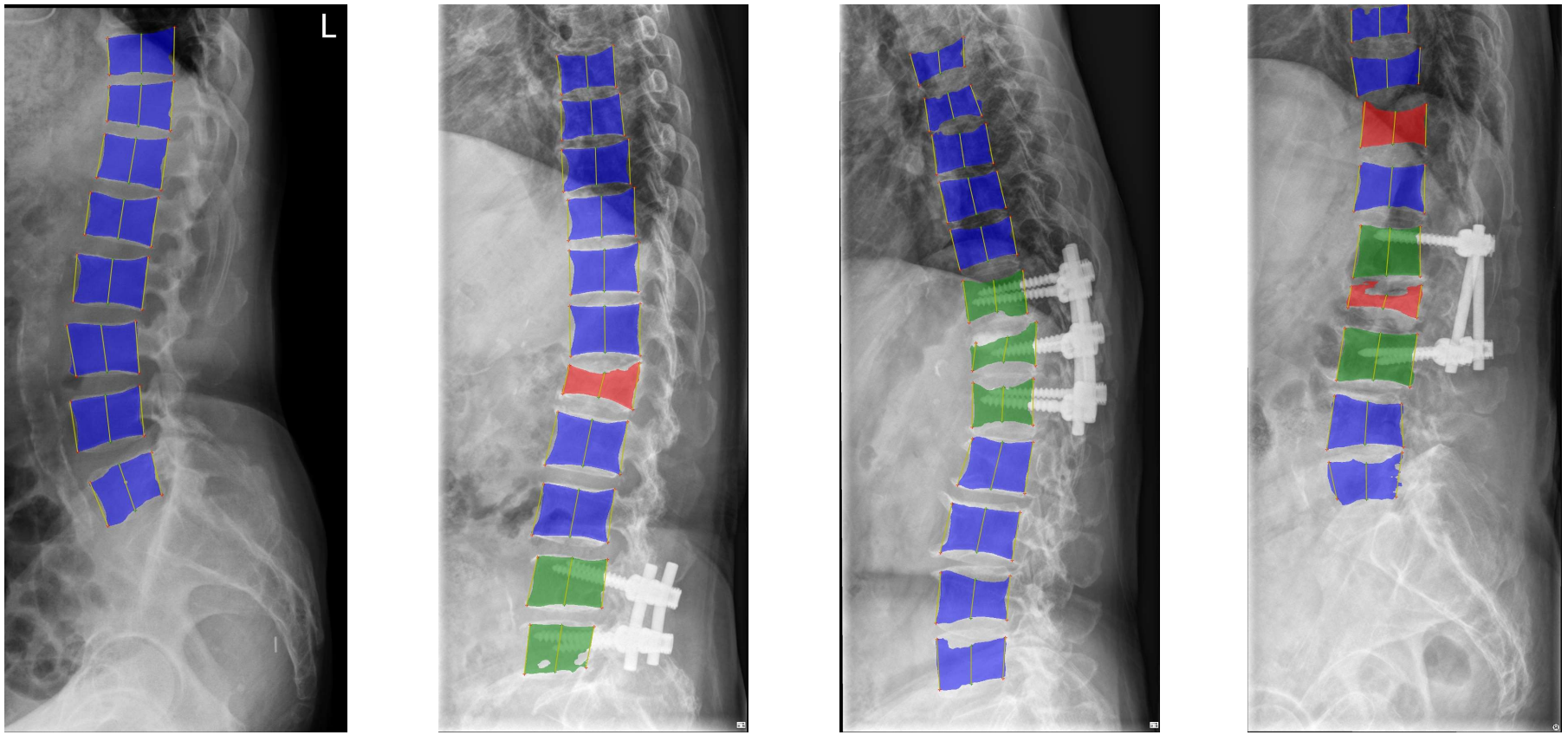
視覺化與執行

- 簡易執行範例
- 選取圖片加入目標資料夾
- 執行指令使用RandomForest
- 目標資料夾名稱rfc

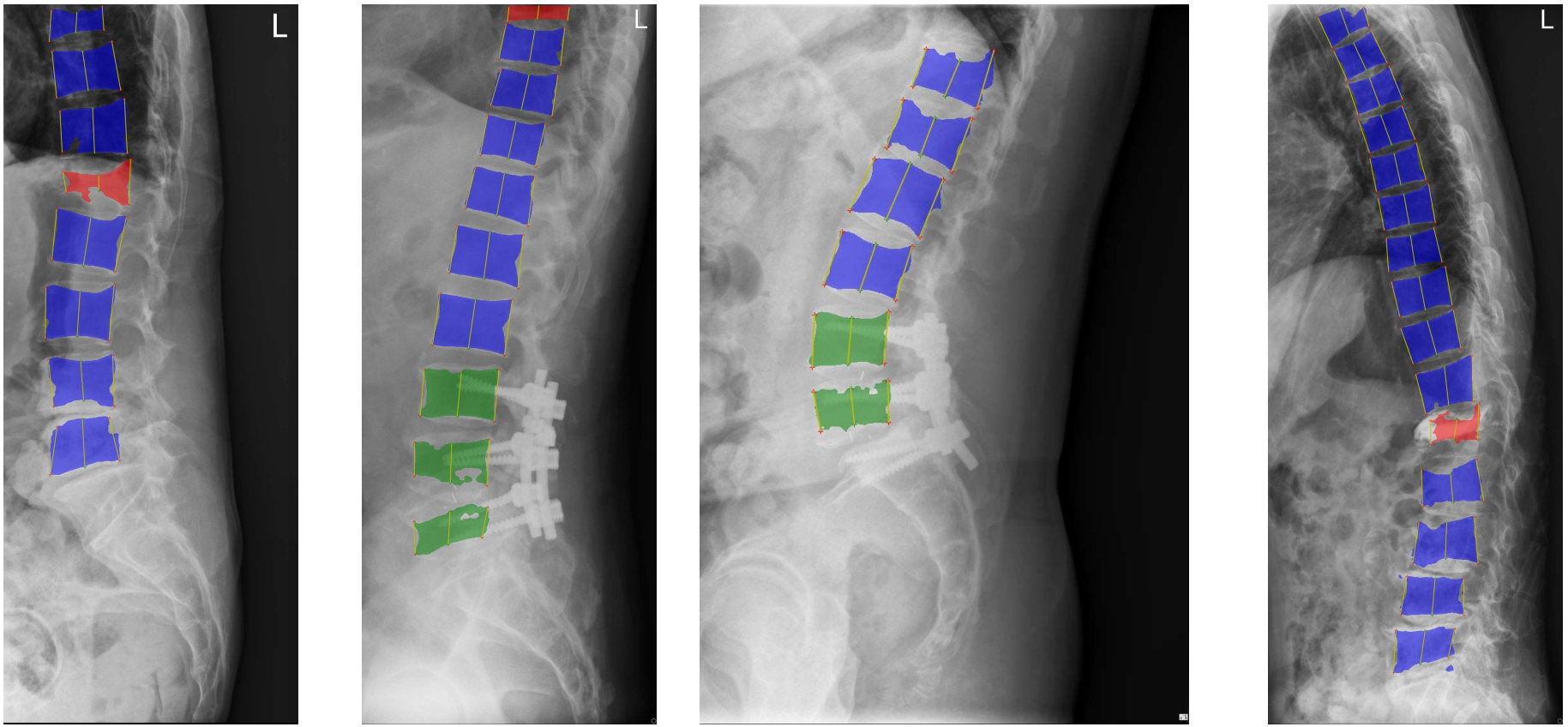



```
PS C:\Users\user\OneDrive\桌面\vertex> python .\Vdetect.py --source C:\Users\user\OneDrive\桌面\vertex\content\DATA\9999999999 --dest C:\Users\user\OneDrive\桌面\
vertex\testResult -me rfc -n rfc
```

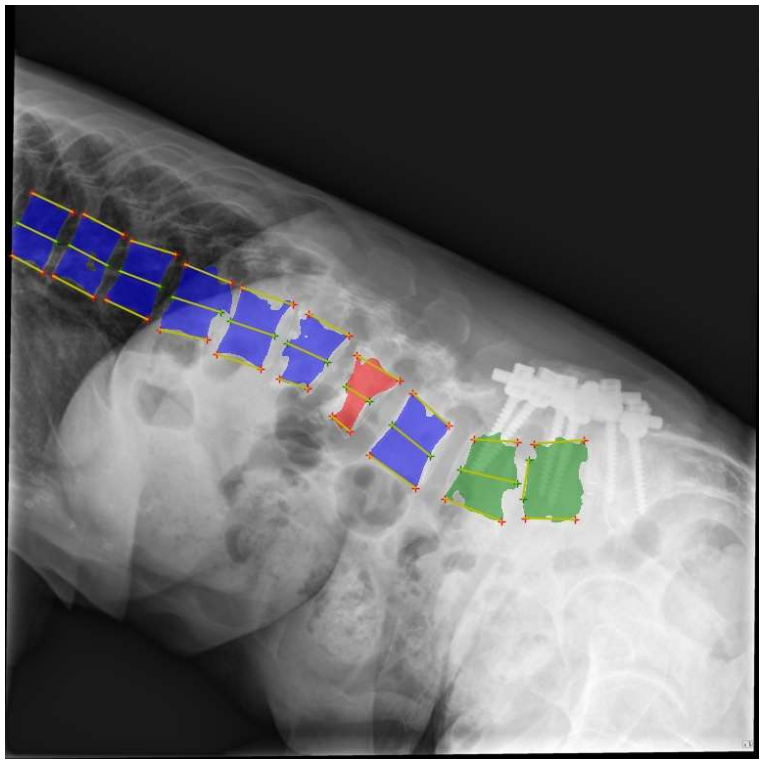
視覺化與執行

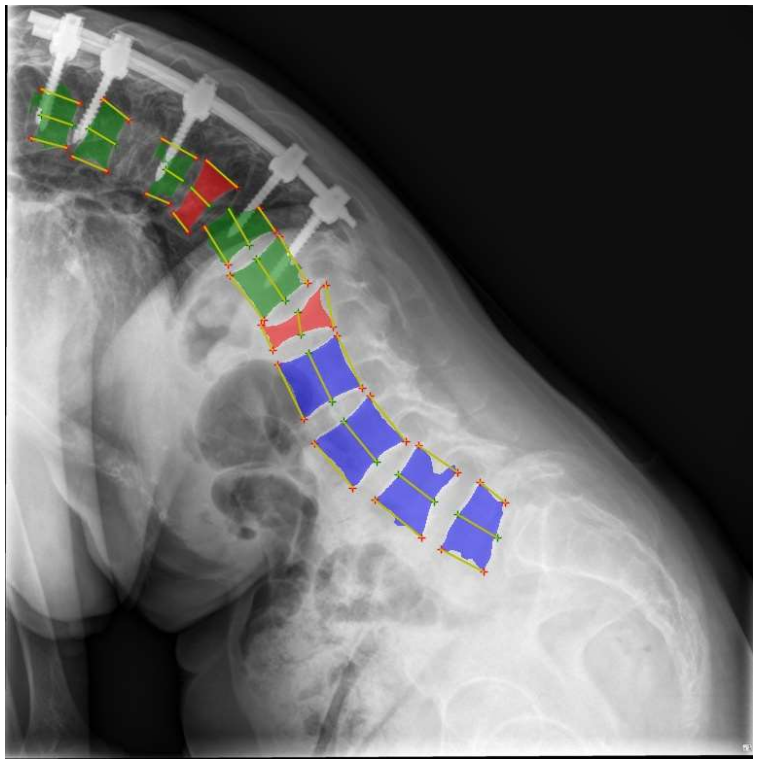


視覺化與執行



視覺化與執行





謝謝聆聽