Group_6 Dark side of the volume

組員: 嚴宇同、林以諾、羅文璟、陳以新、盧柏岑

一、概述

Seismic volumes（地震體積數據）是由地震波勘探技術生成的一種三維數據集，用於展示地下地質結構的詳細資訊。本次期末的目標就是使用此斷層資訊來識別斷層（faults）的位置。使用人工舊方法來分析、標註斷層的位置會花費大量的時間，而且可能會出現錯誤遺漏的部分。現在，我們可以透過機器學習的方法，利用我們訓練出來的模型來自動對斷層數據進行標註。不僅準確率會有所提升，也節省了大量的時間成本。

二、Dataset

Data structure

我們總共有400個資料夾，每個資料夾都包含一張3D地震波結構數據以及 對應的斷層遮罩。3D數據的形狀為(300， 300， 1259)，代表長寬以及地層深度，每個pixel都是浮點數。每張遮罩的維度也是(300, 300, 1259)，不過是 用 (0, 1) 來標記斷層的位置，詳細數據如下:

Seismic Volumes：

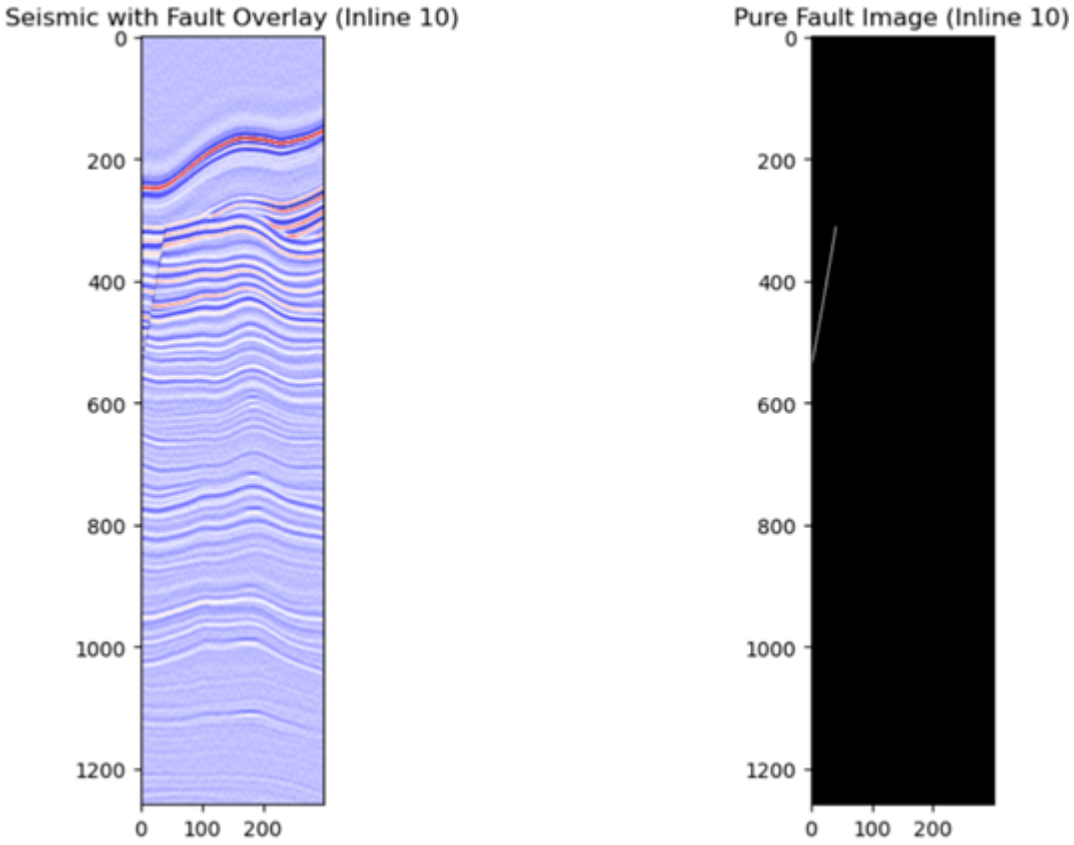
- 表示地層的三維結構數據，通常以三維數組形式存儲。
- 每個數據點的值對應地震波反射的振幅（Amplitude），反映地下材料的物理特性。
- 格式：`(X， Y， Z)` 的三維陣列，其中：
  - X 和 Y 是地表上的橫向和縱向位置。
  - Z 是地層的深度。
  - 每個點的值為浮點數，代表振幅。

Fault Masks：

- 對應地震數據的標註，用來指出每個位置是否存在斷層。
- 格式：`(X， Y， Z)` 的三維布林陣列（或二值陣列），其中：
  - 值為 `1` 表示該位置為斷層。
  - 值為 `0` 表示該位置不是斷層。

Both files are 3D numpy arrays with dimensions (300, 300, 1259).

下圖印出原斷層切面圖以及該切面的斷層標記(Inline 10):



Data preprocessing

2D slice :

每張3D圖片的大小約在850MB左右，相當龐大。雖然三維數據可以提供完整的地下地層結構信息，但直接處理三維數據存在以下挑戰：

1. 計算資源消耗：

   三維數據的計算成本高，無論是內存需求還是運算量，都會顯著增加。

2. 模型設計複雜：

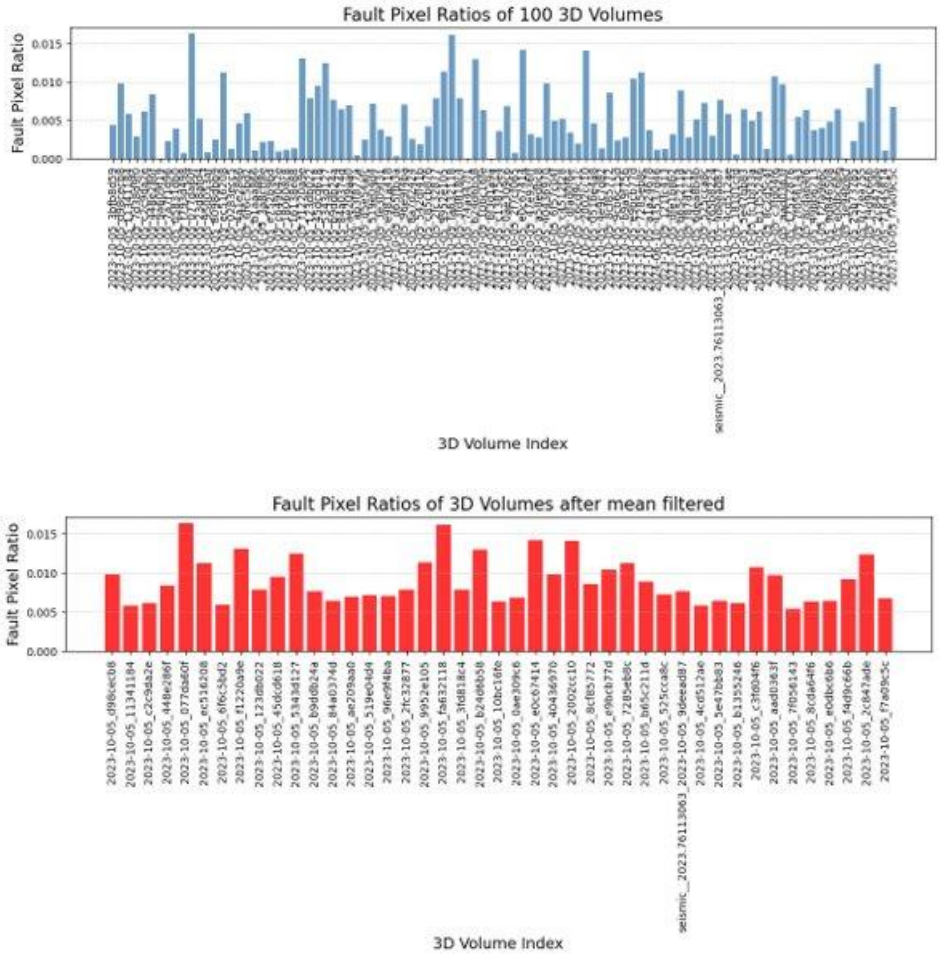
   三維數據要求設計適配的三維神經網絡，這比常見的二維網絡更複雜且訓練難度更大。

因此，我們將 3D 資料沿著 X 軸切片為 2D 切片，從而顯著減少一次性處理的資料量並降低計算資源需求。每個切片都可以獨立處理，使得訓練過程更有彈性。After slicing, both files are 2D array with dimensions(300, 1259).

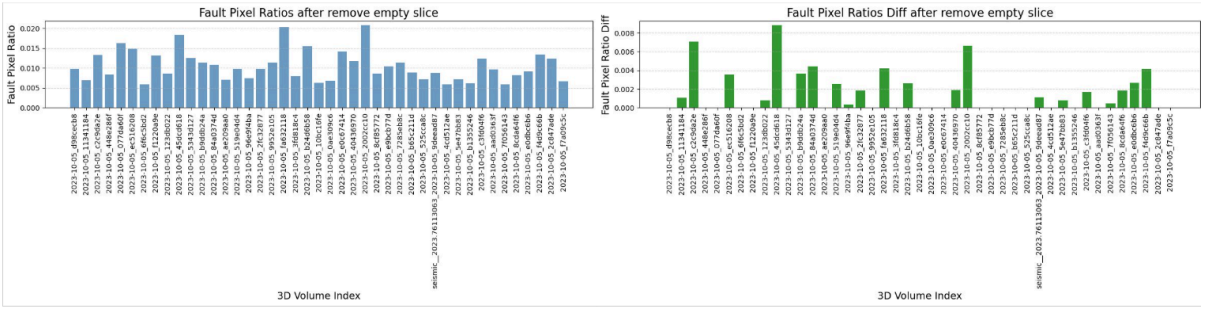
Data preprocessing- filtering

在觀察大部分的斷層標籤之後發現，每張2D切面的斷層佔整張圖片像素非常小的一部份。另外，訓練這麼大量的資料，會花費很多時間，所以我希望能篩選出良好的訓練資料。

第一步驟，我們隨機選擇100張3D地層圖，並計算了所有斷層的平均像素數占比，並濾除了像素數比率在平均值(0.00531)之下的斷層。Fault Pixel Ratio的原始(藍色)與過濾後(紅色)的比較如下圖，可以看出我們選擇了比較有訓練價值的圖片:



第二步驟，我們在做3D切片的同時，又發現許多 2D 切片數據中並不包含任何斷層資訊。我們也濾除了這些沒用的切片。最初的數據集中有 30,000 個切片。經過這兩個步驟處理後，數據集被縮減到 11,441 個切片。下圖左邊是第一步驟後的結果，右邊呈現的是過濾沒有斷層的2D切片後，我們每張圖片的Fault Pixel Ratio提升多寡，提升越大越好:



After removing 2D slices with no faults

Compare Difference

以上這些都在我們自訂義的`LargeSeismicDataset`的`class`裡面一並做處理。

測試DataLoader

```python
# 使用 Dataset 和 DataLoader
dataset = LargeSeismicDataset(data_path=training_data, slice_dim="depth")
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)

# 查看單個 batch
for batch_idx, (images, labels) in enumerate(dataloader):
    print(f"Batch {batch_idx + 1}:")
#     print(images)
    print(f"Images shape: {images.shape}")  # (batch_size, 1, height, width)
    print(f"Labels shape: {labels.shape}")  # (batch_size, 1, height, width)
    break
```

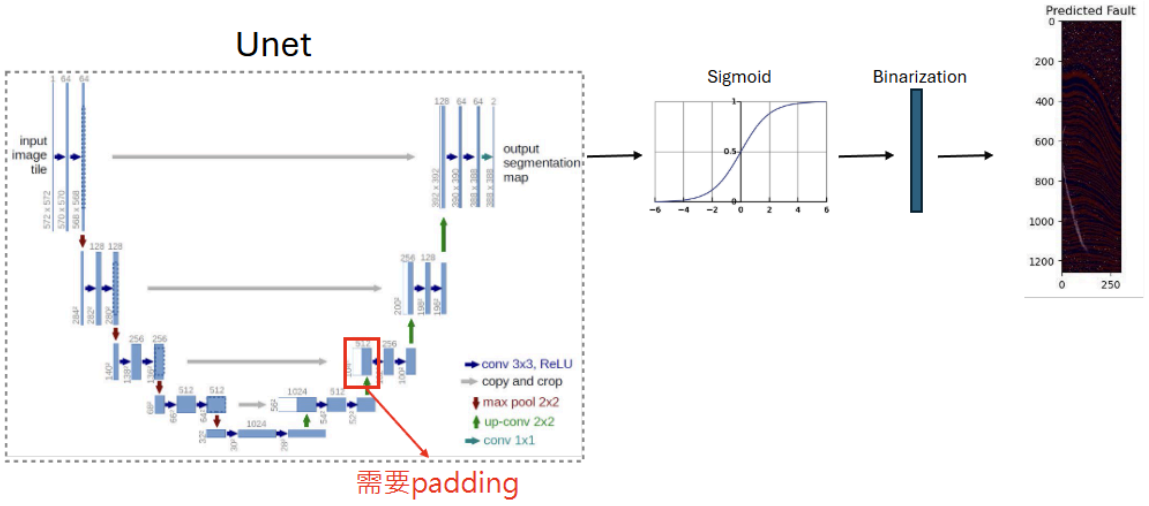
```
Total valid 2D slices after filtering: 11441
Batch 1:
Images shape: torch.Size([4, 1, 300, 1259])
Labels shape: torch.Size([4, 1, 300, 1259])
```

三、模型架構

我們使用U-Net,以下是基於程式碼的模型架構詳細解釋:

模型總體結構

- 收縮路徑(**Contracting Path**):
  - 通過多層卷積和池化操作逐步減小空間維度，同時提取高層次特徵。
  - 每層的輸出被保存，用於後續的跳躍連接(Skip Connection)。
- 擴展路徑(**Expanding Path**):
  - 通過上採樣操作逐步恢復空間維度，同時結合跳躍連接的特徵以保留詳細信息。
- 特徵映射層(**Feature Mapping Layer**):
  - 最後一層使用 1x1 卷積，將特徵映射到所需的輸出通道數。



- Padding: 在skip sonnection 和upsample 的部分有做channel數的修正，在concat的時候檢查兩者通道數是否相同，否則合併會出現錯誤。

```
# 確保 x 和 skip_con_x 的形狀一致
diff_h = skip_con_x.shape[2] - x.shape[2]
diff_w = skip_con_x.shape[3] - x.shape[3]

# 填充 x
if diff_h > 0 or diff_w > 0:
    x = F.pad(x, [0, diff_w, 0, diff_h])  # [left, right, top, bottom]
elif diff_h < 0 or diff_w < 0:
    skip_con_x = F.pad(skip_con_x, [0, -diff_w, 0, -diff_h])  # [left, right, top, bottom]

# 拼接
x = torch.cat([x, skip_con_x], axis=1)
```

- Output with Sigmoid、Binarization

  除了Unet以外，因為Unet的輸出是浮點數並且有負值，我們希望限制輸出介於0和1之間。所以再經過sigmoid，最後再二值化，設定Thershold=0.5把每個值改成0或1。

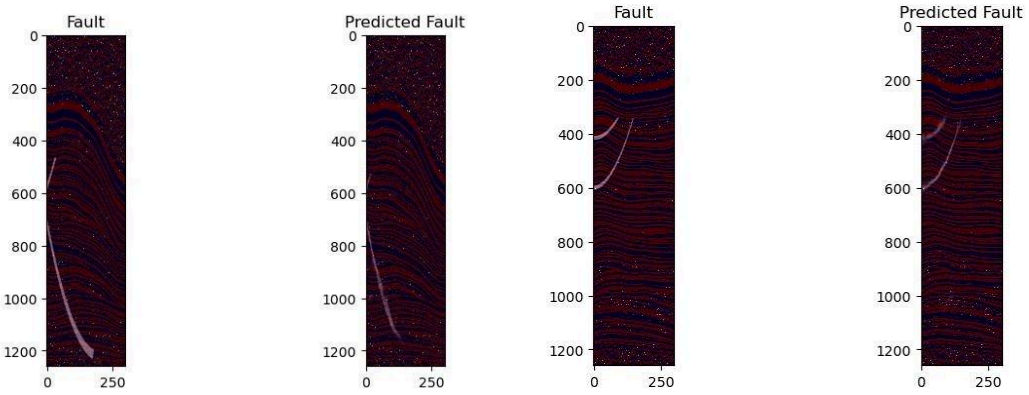
四、學習策略

- Hyperparameter

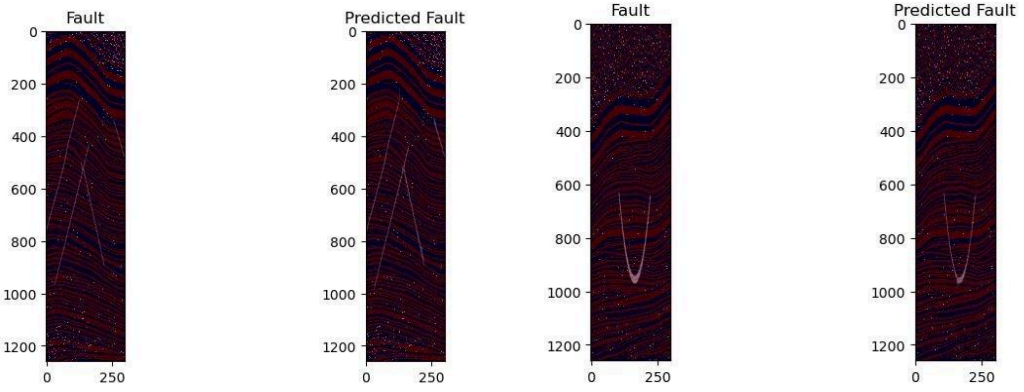
```
# Hyperparameters
# criterion = nn.BCEWithLogitsLoss()
batch_size = 4
n_epochs = 5
input_dim = 1
label_dim = 1
display_step = 50 # 每 50 step印出當前訓練的效果
lr = 0.0002
device = 'cuda'
```

- 訓練參數

  我們訓練5個epoch, loss使用Binary Cross Entropy, Optimizer則是adam。下圖是我們訓練的4個結果, 左邊Fault是真實斷層位置, 右邊Predicted Fault是預測的斷層位置。上方兩張為效果不佳的結果, 下方為預測好的結果。我們發現模型預測出的斷層細節很差, 真實圖片中很粗的斷層, 預測結果邊緣變模糊或是變很細, loss為0.02113以及0.00491



  預測不佳的結果(loss:0.0213)



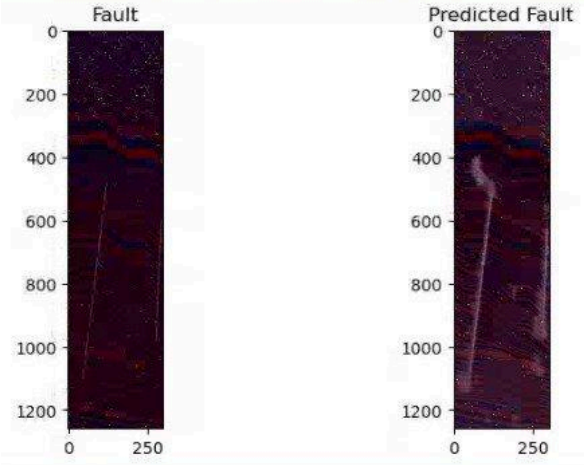
  預測好的結果(loss:0.00491)

- 改進方法1:Weighted BCE loss

  我們觀察到圖片大部分pixel都不是斷層部分(會被分割成0), 所以我們想或許增加斷層部分(被分割為1)在loss中的權重可以改進準確度。

  所以我們為1的部分設定權重 0的pixel數量 / 1的pixel數量

結果如下圖。使用這個加權方法後卻發現原本少少的斷層被過度放大，可能是權重過大導致，因為大部分圖片斷層的占比都非常小，讓訓練結果很不平衡。



- 改進方法2:Tversky loss

因為前面結果不佳，我們上網找資料發現影像分割中很常使用的loss為Tversky loss，可以比較好處理正負樣本懸殊的情況。

Tversky Loss:針對影像分割任務設計的loss，特別適用於類別不平衡的情況。目標物件只佔整體影像的一小部分時，普通的BCE loss可能會對背景過於敏感，導致模型偏向預測背景。Tversky Loss將 False Positives和 False Negatives賦予不同的權重來緩解這個問題。

$$\text{Tversky Index (TI)} = \frac{\text{TP}}{\text{TP} + \alpha \cdot \text{FP} + \beta \cdot \text{FN}}$$

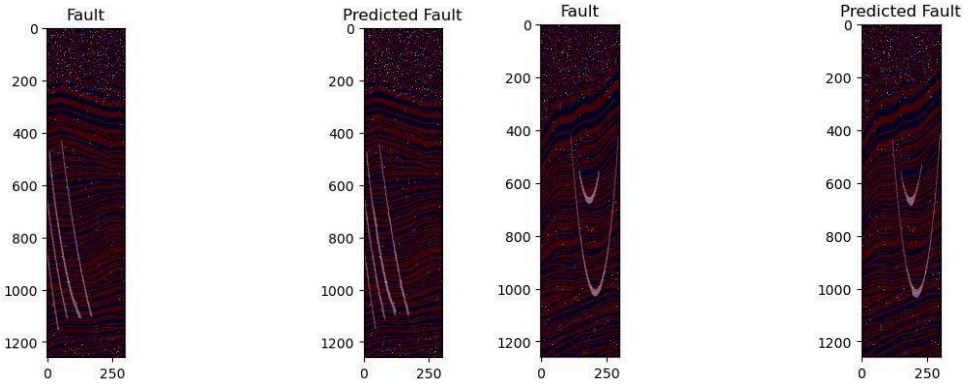
$$\text{Tversky Loss} = 1 - \text{Tversky Index}$$

只要調整α、β值就可以讓模型有不同的處理標準，如下:

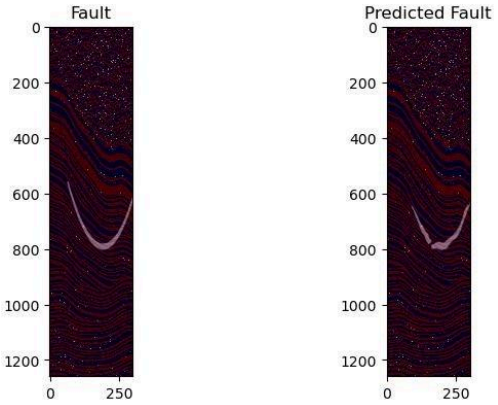
α>β:使模型更保守，更傾向預測0

α<β:使模型更積極，更傾向預測1

我們設定 α=0.7、β=0.3，結果如下圖:



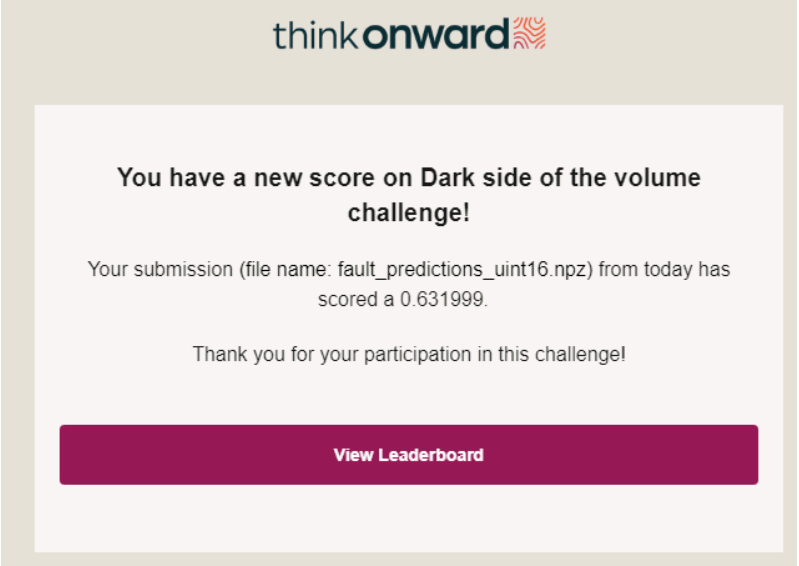
Tversky loss預測好的結果 (loss:0.1654)



Tversky loss下預測不好的結果(loss:0.3258)

可以發現比較沒有預測大幅度出錯的情況，預測結果也更好。並且我們還發現原始BCE loss雖然比較小(0.0213、0.00491)，Tversky loss(0.1654、0.3258)，但Tversky loss表現比較好，所以我們認為Tversky loss比較能表現我們訓練得好壞。

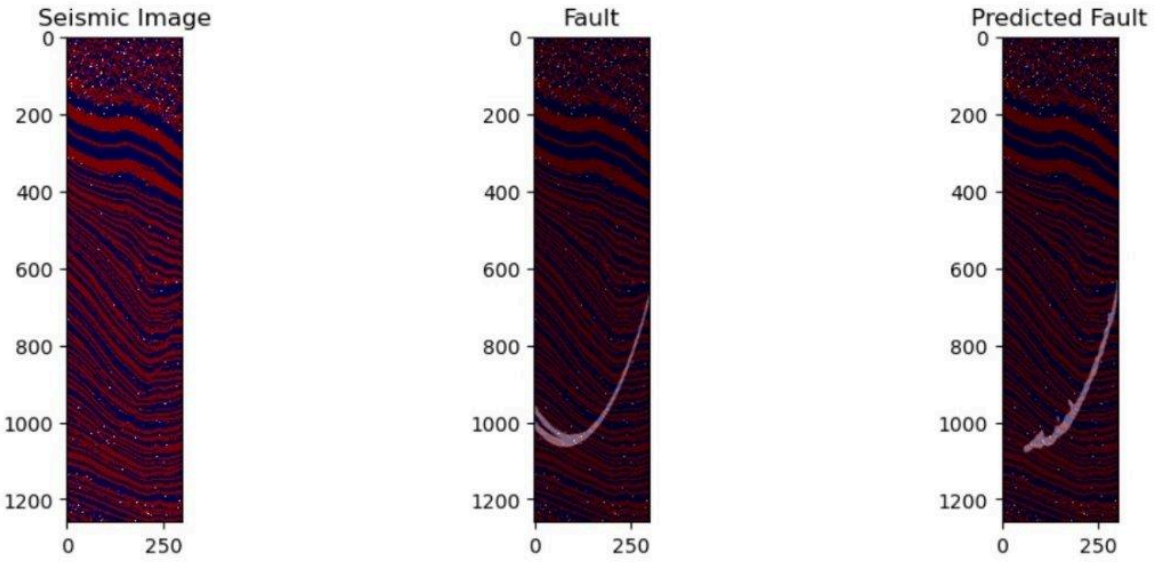
五、結論與討論(final score = 0.631999)



在這次的期末專案中，我們的做法是將3D的訓練資料先沿著x軸切成2D，這樣的做法處理2D數據會比3D數據所需的運算資源少，訓練速度更快；並且許多現有的深度學習模型（如ResNet、EfficientNet）已經針對2D數據不斷進行改良，使用這些預訓練模型可以更快地進入應用階段。此外，若想提高模型的泛化能力，在數據增強的部分對2D切片進行數據增強（如旋轉、裁剪、縮放等）都比3D數據更容易實現。

然而，凡事都是一體兩面的，這樣的處理方式也必然存在許多隱藏的問題。例如將3D數據切分為2D後，x軸上的連續性和關聯性可能會丟失；或是模型在訓練過程中只能學到2D的局部特徵，而無法直接學習3D的全局空間結構，這可能導致性能下降；又或者某些特徵可能存在於多個切片之間，分成2D後將可能無法捕捉這些特徵。

若是未來想改進本次專案的模型，我們認為可以使用跨切片融合的方式，例如在訓練後或組合階段，透過卷積神經網絡（CNN）或基於Transformer的架構，融合多個相鄰切片的資訊；或是改進切片選擇的過程，例如選擇最有代表性的切片或採用多平面切片（多角度視圖），並且引入預處理或後處理步驟，在切分和重組的過程中應用插值技術或基於深度學習的修復方法，以期能減少資訊的丟失。



此外，我們的模型在特定情況下預測誤差較高，顯示出準確率和泛化能力的下降。我們認為可以透過調整Effective Reception Field(ERF)來解決這個問題。當ERF過小，就只能專注於很細微的細節，而無法捕捉長距離依賴無法了解更大範圍的結構；但如果它的ERF過大，會捕捉到太多無關的內容，結果無法聚焦於最重要的細節，導致特徵稀釋，而導致模型的預測效果不佳。透過平衡局部與全局特徵提取，並適應不同數據模式的需求，應該能解決我們模型在特定情況效能不佳的問題。而調整ERF的方法包含融合注意力機制：加入通道與空間注意力模塊 (如 SE、CBAM) 以動態調整 ERF 重要性；或是透過數據增強與正則化：加強數據多樣性，避免模型過於依賴特定 ERF，並應用 dropout 或 mixup 減少overfitting的風險。