U-Net 神經網路的理論研析與應用實作

期中報告/電機博 2 吳杰倫

1. U-Net 設計理念與模型架構

U-Net 是在深度學習推動醫學影像分析發展的背景下，由 Olaf Ronneberger 等人在 2015 年提出的一種神經網路架構[1]。其通過 Encoder-Decoder 結構與跳層連接機制，解決了傳統分割方法的精度與效率問題，廣泛應用於生物醫學影像的分割任務[1]。

U-Net 的模型設計基於 Encoder-Decoder 結構，利用下採樣提取高層次特徵，再通過上採樣逐步恢復空間信息，實現特徵提取與細節復原的結合。跳躍連接（Skip Connections）則有效融合編碼器與解碼器各層的特徵，保留更多細節，提升影像分割的精度與效果。

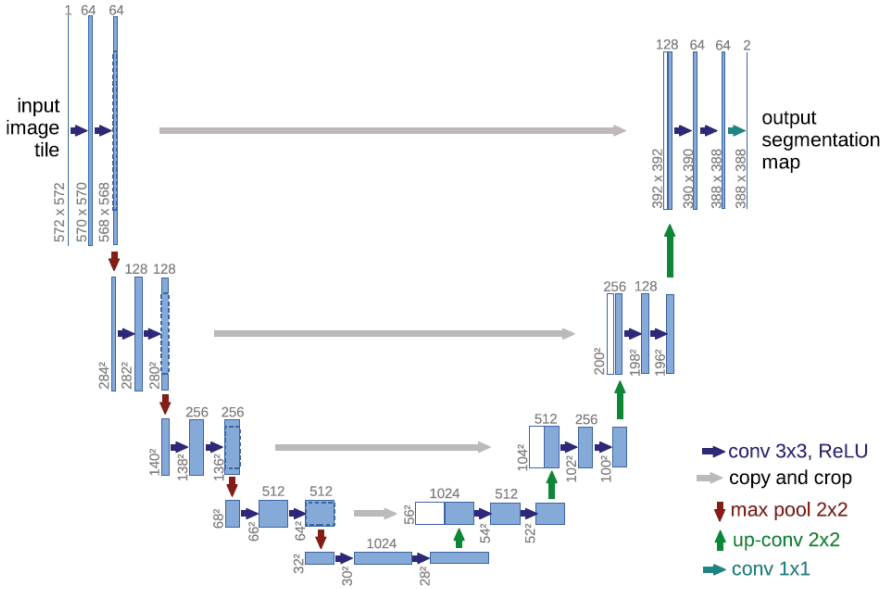


圖 1. U-Net 模型架構[1]

圖 1 是 U-Net 的經典架構，可以分為上採樣與下採樣兩個階段。下採樣階段負責提取低分辨率的深層特徵，用於捕捉圖像的全局信息；而上採樣階段則通過跳躍連接（skip connection）將對應層的高分辨率特徵融合進來，補充細節信息，確保還原精細的邊界和紋理。這種結構設計使模型能夠在不同尺度上同時學習圖像的全局和局部特徵，特別適合像素級的分割任務，如語義分割和醫學影像分割等需要高精度的應用場景。

- **下採樣設計與實作:**

  下採樣的主要目的是捕捉圖像的全局上下文特徵，同時逐步縮小空間維度並增強通道數。在每個下採樣步驟中，首先通過 3×3 卷積層結合 ReLU 激活函數提取局部特徵，隨後使用 2×2 最大池化層將空間維度減半，從而在保留主要特徵的同時顯著降低計算複雜度。經過多次池化操作，特徵圖的尺寸從原始的 572×572 減小至 32×32，但通道數則由 1 增加到 1024，進一步豐富了圖像的表達能力。

- **上採樣設計與實作**：

  上採樣的目的是恢復圖像的空間分辨率，並融合對應下採樣階段的高分辨率特徵。在每個上採樣步驟中，利用 2×2 轉置卷積層（up-conv）將特徵圖進行上採樣，使空間尺寸逐步恢復。同時，通過跳躍連接（copy and crop）將對應的下採樣階段的特徵融合進來，從而保留高分辨率的細節信息，實現精細化的特徵重建，為最終輸出提供更加準確的空間細節和上下文信息。

2. 損失函數設計

二元交叉熵(Binary Cross-Entropy, BCE)與 Dice 是 U-Net 訓練時常用的損失函數。交叉熵損失主要用於衡量預測值和真實標籤之間的差異，透過對每個像素的預測概率進行估計。

$$L_{BCE} = \frac{-1}{N}\sum_{i=1}^{N}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

其中，y_i 是真實標籤，$\hat{y}_i$ 是預測值。

Dice 源於 Dice 相似係數，專門設計用來處理像素級分割，特別適合類別嚴重不平衡的情況（例如小目標分割），可以有效提升小目標的分割效果，但對損失值的梯度較小，可能影響模型的收斂速度。

$$L_{Dice} = 1 - \frac{2\sum_{i=1}^{N} y_i \hat{y}_i}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} \hat{y}_i}$$

從 Dice 的定義中可以看出，如果圖像中需要被標記的區域$(y_i)_{label}$=1，而背景區域設為 0，則$\sum_{i=1}^{N} y_i \hat{y}_i$ 可以被看作需要被分離出來的區域和模型預測區域的交集，當交集越大，L_{Dice} 越小，模型的性能也就越好。

實際訓練時，通常會以某種比例混合 BCE 與 Dice 作為模型的最終損失函數:

$$L \equiv \alpha L_{BCE} + \beta L_{Dice}$$

Dice 項用於引導模型關注整體區域的準確性，而 BCE 則用於改善標註區域細部輪廓的像素類別誤差。

3. U-Net 應用實作

參考 Medium 文章 *Cooking your first U-Net for Image Segmentation* 以及對應的 Github [2][3] 代碼，訓練 U-Net 來對影像進行語意分割(Semantic Segmentation)，資料集來自於 Kaggle Brain MRI Segmentaion 圖集[4]。

A. 軟體開發環境架設

I. 選用 IDE 並安裝 Python

由於目標專案使用 Python 來實現 U-Net，所以本次實作在 Windows 作業系統下，選用微軟發行的 Visual Studio Code 搭配 Python 3.9.13 進行。

II. 建立 Unet 專案根目錄並配置訓練圖集

訓練圖集(https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation) 下載後，壓縮檔內包含 kaggle_3m 與 lgg-mri-segmentation2 個資料架，解壓縮至專案根目錄(Root: C:\Unet)

III. 配置專案依賴項目

原始專案中沒有設置依賴模組的清單，所以需要追蹤原始碼並手動安裝所有需求的模組，以下整理所有需要的模組名稱與版本，在專案根目錄中建立虛擬環境後，利用 pip 指令逐個安裝。

```
# update pip version to 24.3.1
python.exe -m pip install --upgrade pip==24.3.1
# install dependancy module
pip install numpy==1.26.3
pip install pandas==2.2.3
pip install opencv-python==4.10.0.84
pip install matplotlib==3.9.1.post1
pip install albumentations==1.4.21
pip install scikit-learn==1.5.2
pip install tqdm==4.67.0
```

IV. 安裝 Pyotch

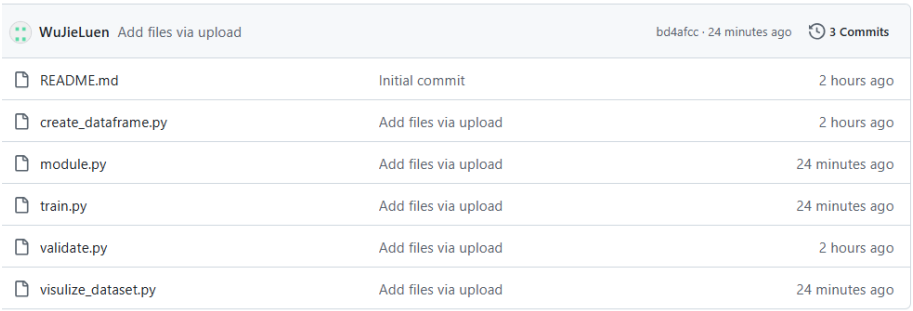
```
pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 --index-url https://download.pytorch.org/whl/cu117
```

由於原始專案沒有提供 U-Net 模型檔，所以需要一張可以支援 CUDA 的 Nvidia 獨立顯示卡並安裝 GPU 版本的 Pytorch 進行訓練。

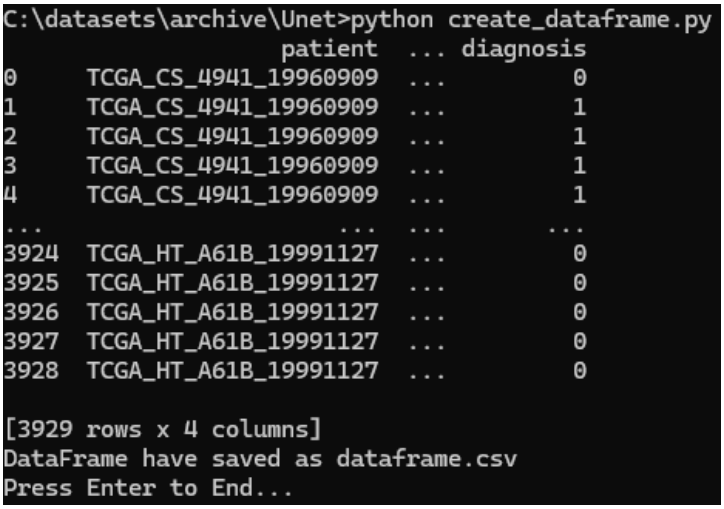
使用 GPU 版本的 Pytorch 需要安裝 Nvidia 的 CUDA toolkit [5]。需要注意的是電腦顯卡的型號、Pytorch、CUDA 與 Python 的版本彼此互相綁定，需要事先確認工具鍊的所有版本彼此相容，本次實驗版本配置組合:Python: 3.9.13、Pytorch: 2.0.1、CUDA: 11.7

B. 建立數據集

I. 原始 github 專案中用來連結圖檔、mask 圖檔與腫瘤診斷信息的標註檔案(data.csv)無法對應到最新版的數據集，所以修改 github 中的代碼重新建立 data.csv，代碼放置在個人 Github 倉庫中。(https://github.com/WuJieLuen/Unet)。

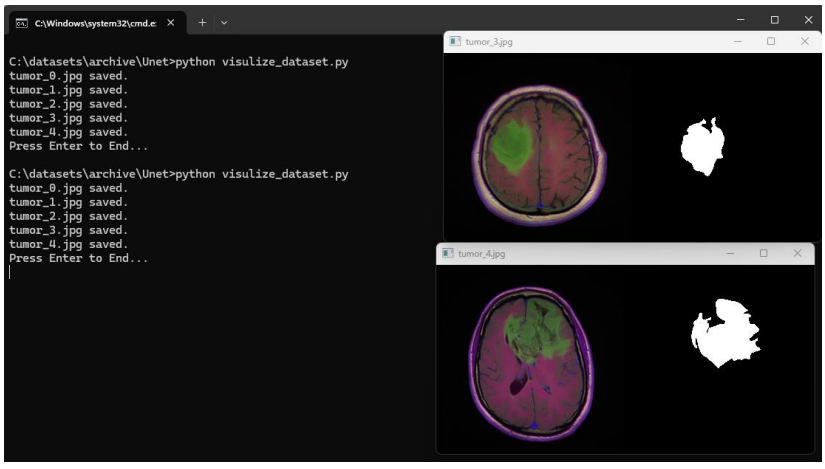


II. 執行 create_dataframe.py 可以生成對於最新圖集的 dataframe.csv



III. 執行 visulize_dataset.py

這個腳本用於查看 dataframe.csv 中的標註資訊與原始圖檔是否匹配，同時可以看到圖檔中的大腦 MRT 影像與腫瘤位置的 Mask 標記。



C. 代碼除錯

由於 Medium 中原作者在 github 代碼有問題，需要修改後才可運行，主要的錯誤為計算 BCE 與 Dice 損失時，沒有將 ground 的 mask data 正規化到[0,1]範圍，原作者將 ground truth tenso 以[0,255]的數據範圍輸入損失函數中計算導致梯度計算異常。另外在 Dice 損失計算中(def dice_coef_loss())，錯誤地使用.sum()方法，使得數據在 batch 方向相加，這並不符合 dice 計算的原則。在這些錯誤的影響下，最後訓練出來的模型預測會完全失準，經過修正後的損失函數放置在 module.py 檔案中。

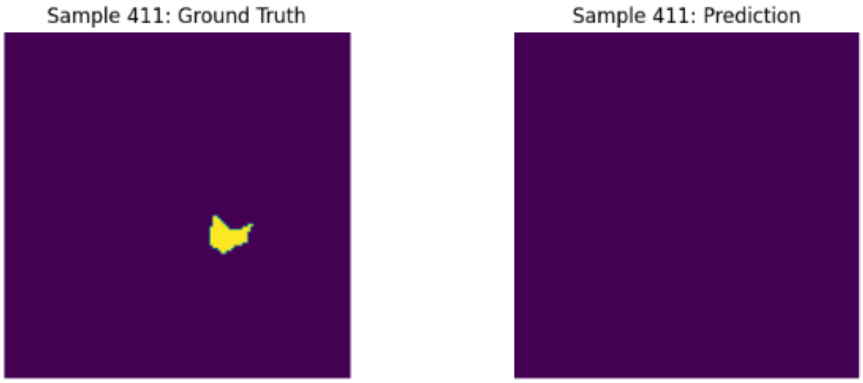


圖 2 原作者 github 模型預測結果圖

D. 代碼優化

I. 優化模型架構:

- ✓ 在卷積層後添加 BatchNormalization 層來穩定訓練過程，避免梯度消失/爆炸
- ✓ 將激活函數從 Relu -> LeakyRelu(alpha = 0.1) 加快學習速度
- ✓ 調整 UNET 建構方式，可以透過參數控制 UNET 規模，方便調適，base = 4、8、16、32、64。

II. 優化學習策略:

- ✓ 根據 Loss 量級調整 back propagation 步長
- ✓ 根據 Loss Profile 建立 Early Stop 機制
- ✓ 改變優化器 Adamax -> AdamW

III. 修改 github 源碼，優化工作流程。

- ✓ 將資料夾的建立、模型訓練與模型測試從單一腳本檔案中分開
- ✓ 在訓練過程中，根據當前 Loss 儲存最佳模型，並建立 checkpoints 機制，每隔固定 epochs 保存模型與優化器狀態，方便接續訓練。

E. 模型架構

I. 編碼器（下采樣路徑）

特徵提取：使用 double_conv 函數進行兩次卷積操作，主要參數為：

- ✓ conv_down1: 輸入通道數為 3（RGB 圖像），輸出通道數為 64。
- ✓ conv_down2: 將特徵進一步處理，輸入通道數為 64，輸出通道數為 128。
- ✓ conv_down3: 輸入通道數為 128，輸出通道數為 256。
- ✓ conv_down4: 輸入通道數為 256，輸出通道數為 512。
- ✓ 下采樣：使用 MaxPool2d 進行 2 倍下采樣。

II. 解碼器（上采樣路徑）

- ✓ 上采樣：使用 Upsample 以雙線性插值方式將特徵圖上採樣 2 倍。

特徵融合與卷積：

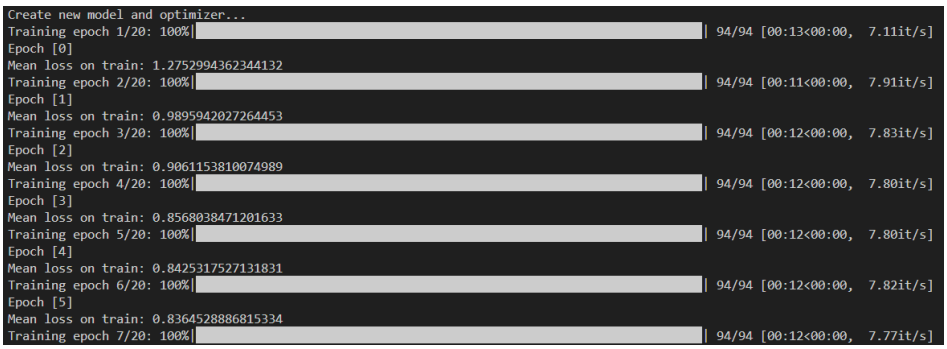
- ✓ conv_up3: 將上層輸出的特徵（512 通道）與對應編碼器層的跳躍連接特徵（256 通道）拼接，輸入 768 通道，輸出 256 通道。
- ✓ conv_up2: 拼接後輸入 384 通道（256+128），輸出 128 通道。
- ✓ conv_up1: 拼接後輸入 192 通道（128+64），輸出 64 通道。

III. 最終輸出層

使用 last_conv 卷積層將通道數降到 1，輸出尺寸與輸入圖像一致，用於二分類（腫瘤/非腫瘤）

F. 模型訓練與測試

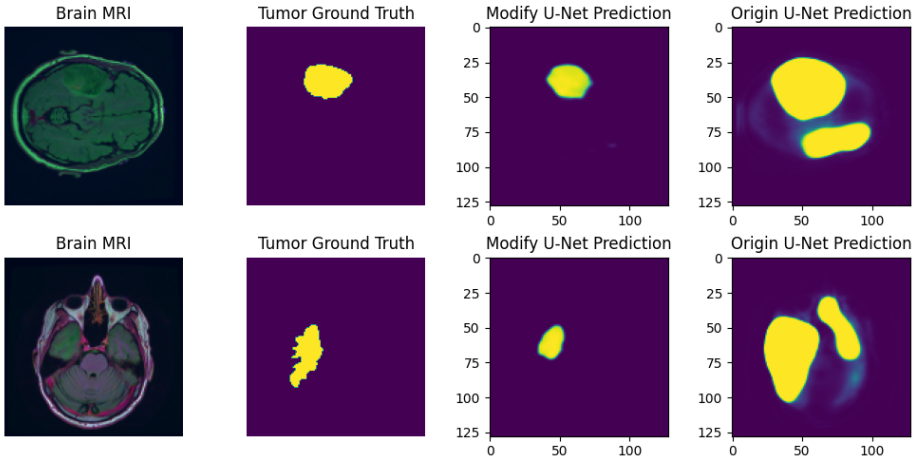
train.py 腳本使用 Adam 優化器，學習率= 0.001，Epoch = 20，Batch = 32 並將數據集分割為訓練集、驗證集與測試集，訓練過程如下:



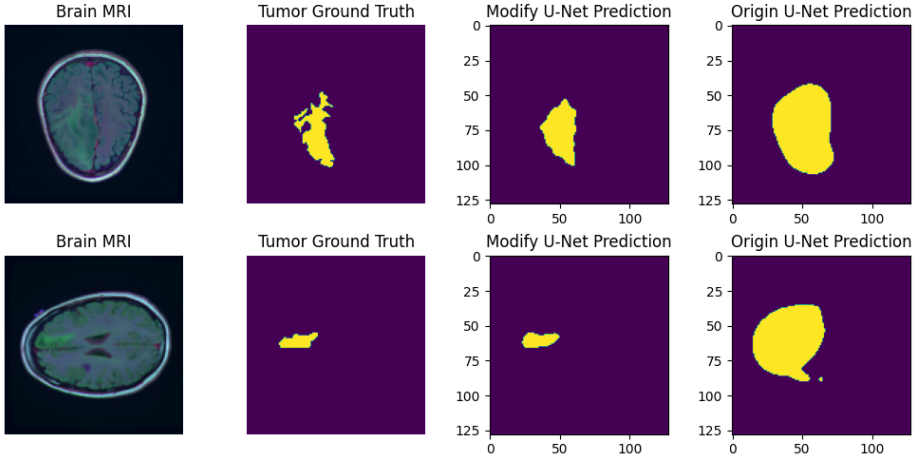
注: 以 GPU 運算時每秒可以計算約 8 次，以 CPU 計算時每 1 次~3 秒

運行 validate.py ，在同樣經歷 20 epochs 的情況下，將原始圖像、腫瘤位置 Ground Truth、修正後模型預測、原始專案的預測結果繪圖比較:

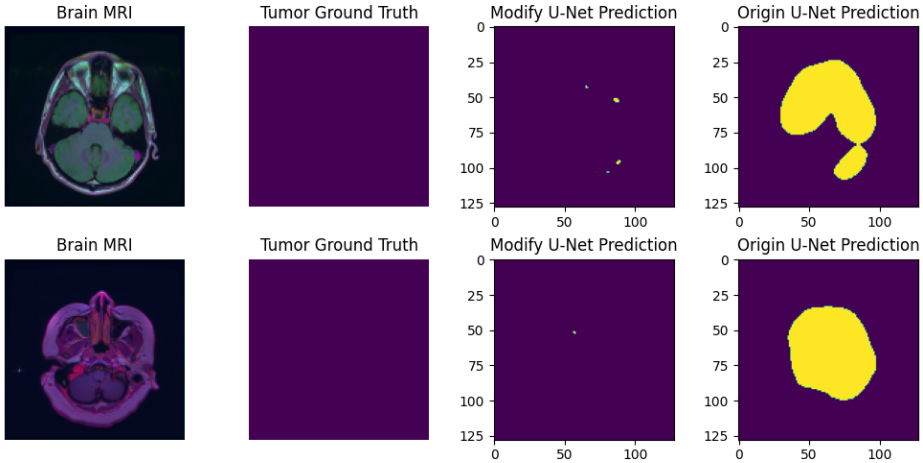
Brain MRT 1 (Tumor)



Brain MRT 2 (Tumor)



Brain MRT 3(Sane)



4. U-Net 實驗心得與結論

- U-Net 預測結果顯示，利用跳躍連接來融合不同解析度的特徵確實可以在複雜的圖像中實現物件的分割，在 Brain MRT1 與 Brain MRT2 的 Tumor 影像中，U-Net 網絡成功預判了腫瘤的位置，但形狀不夠完美，在 Brain MRT 3 的正常大腦影像中，U-Net 的預測雖然存在誤判，但區域不大。
- 原始 Github 專案存在許多錯誤與不合理的設計，使用原始代碼無法順利完成模型訓練，即便修正錯誤之後，也無法得到好的訓練結果，從 Brain MRT 1~3(右 4)的影像中可以觀察到修正前的模型預測結果完全失準。
- 修正後的模型(Brain MRT 1~3，右 3) 在細部輪廓尚無法很好的吻合 Ground Truth，這與 Loss 函數中 BCE 與 Dice 的混合比例有關，因為 Dice 損失主要負責腫瘤整體的位置預測誤差，而 BCE 可以評估像素等級的分類誤差，透過調整 BCE 與 Dice 的比例可以控制 U-Net 最終預測的效能，BCE 比重大，則模型更傾向於學習細節，Dice 比重大模型則傾向於學習腫瘤在大腦中的位置。

5. 參考資料

[1] Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. In International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI).

[2] https://medium.com/@francoisporcher/cooking-your-first-u-net-for-image-segmentation-e812e37e9cd0

[3] Github Project, https://github.com/FrancoisPorcher/awesome-ai-tutorials/tree/main/Computer%20Vision/UNet

[4] https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation

[5] https://developer.nvidia.com/cuda-toolkit