1. Model Architecture

我一開始是使用簡單的 Unet，後來採用 Unet++的架構，下方是我最後一次實驗用的模型架構:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 256, 256]	9,408
BatchNorm2d-2	[-1, 64, 256, 256]	128
ReLU-3	[-1, 64, 256, 256]	0
MaxPool2d-4	[-1, 64, 128, 128]	0
Conv2d-5	[-1, 256, 128, 128]	16,384
BatchNorm2d-6	[-1, 256, 128, 128]	512
ReLU-7	[-1, 256, 128, 128]	0
Conv2d-8	[-1, 256, 128, 128]	18,432
BatchNorm2d-9	[-1, 256, 128, 128]	512
ReLU-10	[-1, 256, 128, 128]	0
Conv2d-11	[-1, 256, 128, 128]	65,536
BatchNorm2d-12	[-1, 256, 128, 128]	512
Conv2d-13	[-1, 256, 128, 128]	16,384
BatchNorm2d-14	[-1, 256, 128, 128]	512
ReLU-15	[-1, 256, 128, 128]	0
Bottleneck-16	[-1, 256, 128, 128]	0
Conv2d-17	[-1, 256, 128, 128]	65,536
BatchNorm2d-18	[-1, 256, 128, 128]	512
ReLU-19	[-1, 256, 128, 128]	0
Conv2d-20	[-1, 256, 128, 128]	18,432
BatchNorm2d-21	[-1, 256, 128, 128]	512
ReLU-22	[-1, 256, 128, 128]	0
Conv2d-23	[-1, 256, 128, 128]	65,536
BatchNorm2d-24	[-1, 256, 128, 128]	512
ReLU-25	[-1, 256, 128, 128]	0
Bottleneck-26	[-1, 256, 128, 128]	0
Conv2d-27	[-1, 256, 128, 128]	65,536
BatchNorm2d-28	[-1, 256, 128, 128]	512
ReLU-29	[-1, 256, 128, 128]	0
Conv2d-30	[-1, 256, 128, 128]	18,432
BatchNorm2d-31	[-1, 256, 128, 128]	512
ReLU-32	[-1, 256, 128, 128]	0
Conv2d-33	[-1, 256, 128, 128]	65,536
BatchNorm2d-34	[-1, 256, 128, 128]	512
ReLU-35	[-1, 256, 128, 128]	0
Bottleneck-36	[-1, 256, 128, 128]	0
Conv2d-37	[-1, 512, 128, 128]	131,072
BatchNorm2d-38	[-1, 512, 128, 128]	1,024
ReLU-39	[-1, 512, 128, 128]	0
Conv2d-40	[-1, 512, 64, 64]	73,728
BatchNorm2d-41	[-1, 512, 64, 64]	1,024
ReLU-42	[-1, 512, 64, 64]	0
Conv2d-43	[-1, 512, 64, 64]	262,144
BatchNorm2d-44	[-1, 512, 64, 64]	1,024
Conv2d-45	[-1, 512, 64, 64]	131,072
BatchNorm2d-46	[-1, 512, 64, 64]	1,024
ReLU-47	[-1, 512, 64, 64]	0
Bottleneck-48	[-1, 512, 64, 64]	0
Conv2d-49	[-1, 512, 64, 64]	262,144
BatchNorm2d-50	[-1, 512, 64, 64]	1,024

```
          Conv2d-344        [-1, 1024, 32, 32]       2,098,176
          Conv2d-345        [-1, 1024, 32, 32]      18,874,368
     BatchNorm2d-346        [-1, 1024, 32, 32]           2,048
            ReLU-347        [-1, 1024, 32, 32]               0
          Linear-348                  [-1, 64]          65,536
            ReLU-349                  [-1, 64]               0
          Linear-350                 [-1, 1024]          65,536
         Sigmoid-351                 [-1, 1024]               0
         SEBlock-352        [-1, 1024, 32, 32]               0
       ConvBlock-353        [-1, 1024, 32, 32]               0
        Upsample-354        [-1, 1024, 64, 64]               0
          Conv2d-355         [-1, 512, 64, 64]         524,800
          Conv2d-356         [-1, 512, 64, 64]       4,718,592
     BatchNorm2d-357         [-1, 512, 64, 64]           1,024
            ReLU-358         [-1, 512, 64, 64]               0
          Linear-359                  [-1, 32]          16,384
            ReLU-360                  [-1, 32]               0
          Linear-361                 [-1, 512]          16,384
         Sigmoid-362                 [-1, 512]               0
         SEBlock-363         [-1, 512, 64, 64]               0
       ConvBlock-364         [-1, 512, 64, 64]               0
        Upsample-365        [-1, 512, 128, 128]              0
          Conv2d-366        [-1, 256, 128, 128]        131,328
          Conv2d-367        [-1, 256, 128, 128]      1,179,648
     BatchNorm2d-368        [-1, 256, 128, 128]            512
            ReLU-369        [-1, 256, 128, 128]              0
          Linear-370                  [-1, 16]          4,096
            ReLU-371                  [-1, 16]              0
          Linear-372                 [-1, 256]          4,096
         Sigmoid-373                 [-1, 256]              0
         SEBlock-374        [-1, 256, 128, 128]              0
       ConvBlock-375        [-1, 256, 128, 128]              0
        Upsample-376        [-1, 256, 256, 256]              0
          Conv2d-377         [-1, 64, 256, 256]         16,448
          Conv2d-378         [-1, 64, 256, 256]         73,728
     BatchNorm2d-379         [-1, 64, 256, 256]            128
            ReLU-380         [-1, 64, 256, 256]              0
          Linear-381                   [-1, 4]            256
            ReLU-382                   [-1, 4]              0
          Linear-383                  [-1, 64]            256
         Sigmoid-384                  [-1, 64]              0
         SEBlock-385         [-1, 64, 256, 256]              0
       ConvBlock-386         [-1, 64, 256, 256]              0
          Conv2d-387         [-1, 13, 256, 256]            845
================================================================
Total params: 114,536,525
Trainable params: 114,536,525
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.00
Forward/backward pass size (MB): 4810.53
Params size (MB): 436.92
Estimated Total Size (MB): 5250.45
----------------------------------------------------------------
```

中間層數太多故先省略，總共有 387 層。

2. Problems and Solutions

一開始遇到的問題是單純用 Unet 的 Performance 沒有很好，不管我怎麼訓練怎麼換其他的 backbone，Test 的結果最好 IoU 只能到 0.7 上下，後來我到網路上參考一些資料有採用 Unet++，這個架構能將更多的特徵傳到 decoder，調整完模型架構後 IoU 可達到 0.8 以上，接著我就開始想一些有沒有方法能繼續提升 Performance，後來試試 SEBlock、CRF(隨機場)、Ensemble，最後 IoU 有達到 0.9。

3. Methods to improve accuracy

以下是我嘗試的策略:

(1) 模型架構

Unet -> Unet++ -> Unet++Improved(SEBlock)

Encoder: ResNet50 -> ResNet101 -> ResNet152 -> ResNeXt101(64x4d) -> ResNeXt101(32x8d)

(2) Loss Function 調整

Cross Entropy -> Cross Entropy + Dice Loss -> Focal Loss + Dice Loss

(3) 資料增強

隨機色調調整、對比度調整

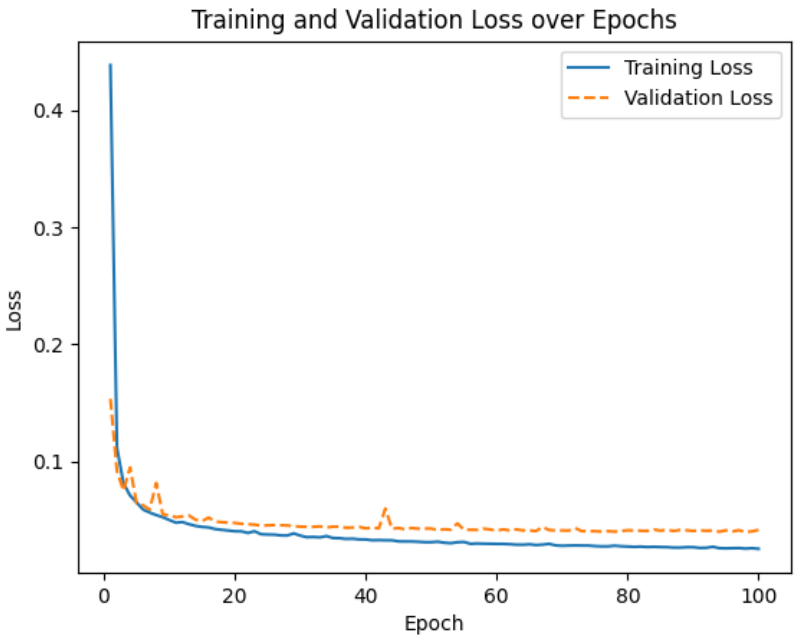
(4) 其他

CRF: 將推理結果使用隨機場演算法處理雜訊

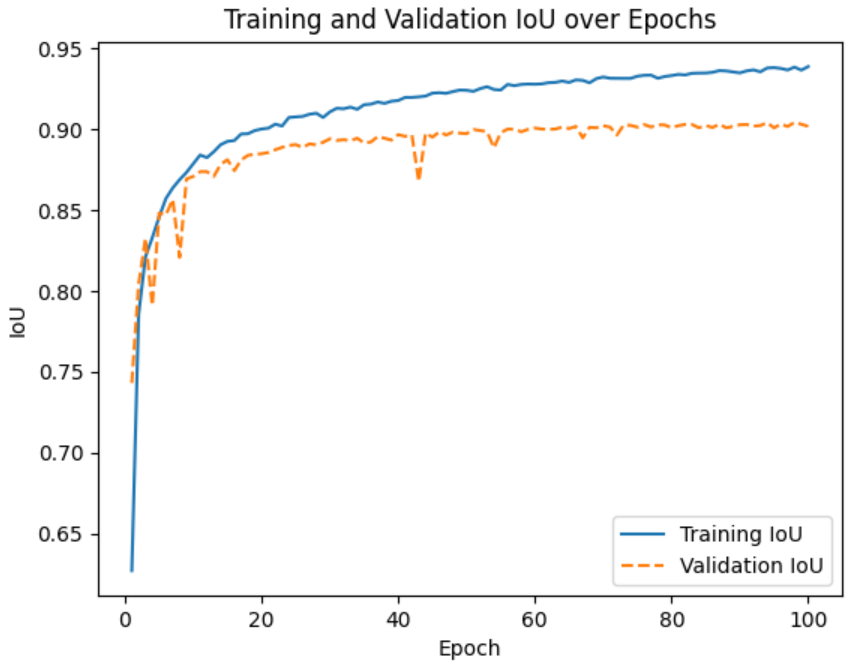
Ensemble: 用訓練過的幾個模型做像素投票得到新的推理結果

4. Analysis of Results

最後一次訓練模型的 Training and Validation Loss 圖



最後一次訓練模型的 Training and Validation IoU 圖



訓練結果:

Date	10 月 26 日	10 月 27 日	10 月 27 日	10 月 28 日	10 月 29 日
Encoder	ResNet16	RestNet50	DenseNet201	ResNet50	ResNet101
epochs	200	300	500	100	200
batch_size	16	16	8	8	8
data_augmentation	no	no	no	no	no
loss function	CrossEntory	CrossEntory	DiceLoss + CrossEntropy	CrossEntory	DiceLoss + CrossEntropy
Type	Unet	Unet	Unet	Unet++	Unet++
Resize	256	256	256	512	512
Test Score	0.67434	0.70367	0.67867	0.84004	0.85312

10 月 31 日	11 月 1 日	11 月 3 日	11 月 4 日
ResNet152	ResNet152	ResNeXt101(64x4d)	ResNeXt101(32x8d)
200	200	100	100
8	8	2	2
no	yes	no	no
DiceLoss + CrossEntropy	DiceLoss + FocalLoss	DiceLoss + FocalLoss	DiceLoss + FocalLoss
Unet++	Unet++	Unet++Improved	Unet++Improved
512	512	768	768
0.85838	0.85791	0.89737	0.89604

特殊方法				
Date	11 月 2 日	11 月 3 日	11 月 4 日	11 月 6 日
	ensemble(Unet++)	Unet++I(CRF)	ensemble(Unet++Improved)	ensemble(Unet++Improved)
Test Score	0.86627	0.86717	0.88291	0.90026

結論:

根據我實驗下來的結果，能夠有效提升 Performance(IoU)的方法主要是修改模型架構還有 Resize 圖片時要大一點，模型越大看起來效果越好，放入模型的圖片越大效果越好，我推測可能是因為的保留的特徵較多，另外 Loss Function 和資料增強看起來對 Performance 的影響有限，可能是 4000 多筆資料其實對 Segmentation 任務來說其實算很多，模型學習的話確實都可以往某個特定的方向達到局部的極值，最後 CRF 演算法和 Ensemble 的像素投票機制可以讓 Performance 提升一些，算是能讓整體效果稍微上升的最終手段。

程式架構說明:

* main.py - 程式進入點
* train.py - 執行訓練、驗證、保存權重還有繪製過程的圖
* dataset.py - 處理和讀取訓練資料
* utils.py - 定義不同的 Loss Function
* Unet.py - 模型架構
* UnetPlusPlus.py - 模型架構
* UnetPlusPlusImproved.py - 模型架構