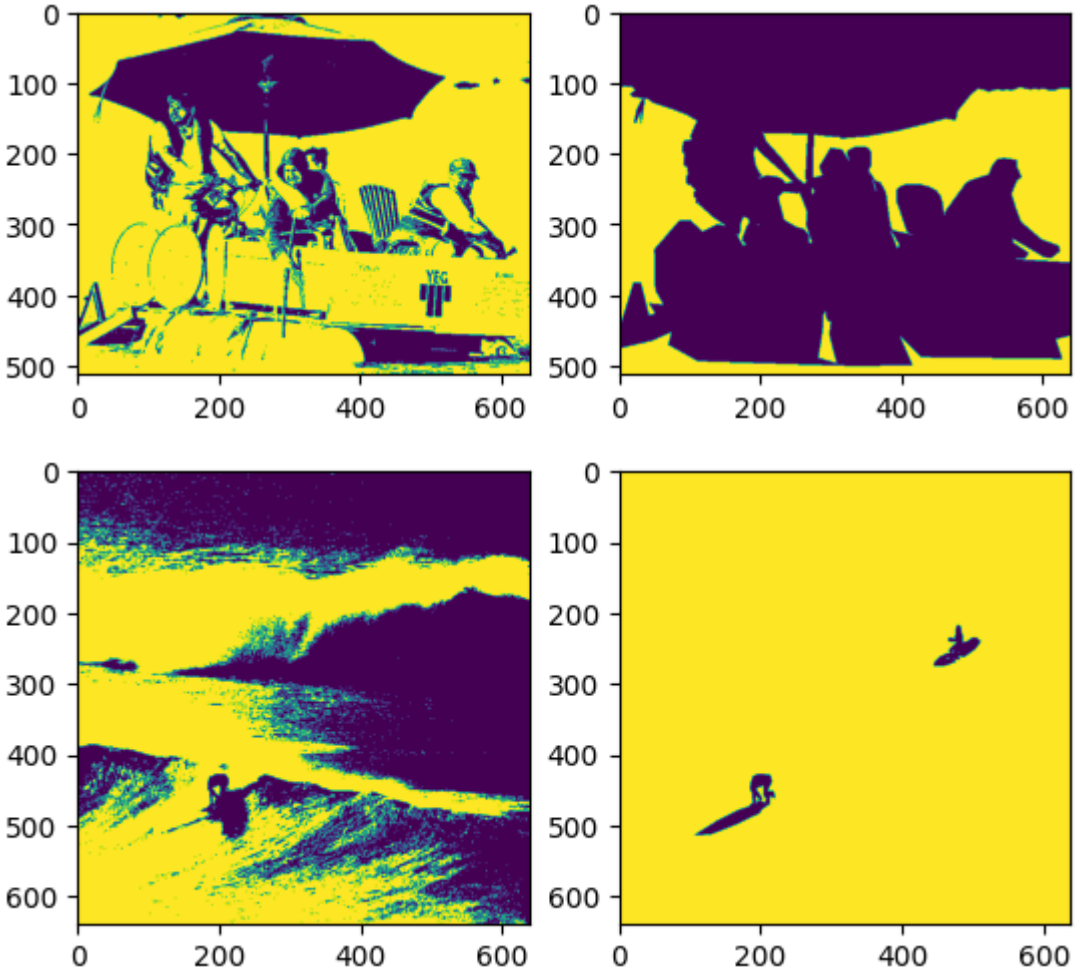
DIP Final report : Water Segmentation

(Deep Learning-based Method)

第三組：電子所 張鈞瑋 313510136 ， 智能所 鄭幃澤 313581040

1. Introduction :

在一開始我們首先嘗試使用傳統方法來實作這次的 Project，所採用的方法包含 Threshold、K-Means 等常見的傳統影像分割方法，但最後平均 IOU 幾乎只有 0.2~0.3 左右，效果非常不好。從下方輸出結果可以看出來，在具有複雜背景的影像中(例如有浪花的海面等等)，傳統方法很難正確分割影像(右邊為 Ground Truth、左邊為 Prediction)。



因此，我們選擇改用深度學習而非傳統方法來進行 Water Segmentation，其原因如下：

(1) **特徵自動提取**：傳統方法需要設計一套合適的特徵提取過程，這通常需要大量的時間來調整特徵工程的細節。而深度學習可以自動從數據中學習有效的特徵，尤其像是 CNN 可以自動學習顏色、形狀等多層次特徵。

(2) **效能比較**：傳統方法的效能受限於手工設計的特徵與分類演算法的簡單性，比較難以應對複雜背景與不同光照條件下的水體分割。而深度學習模型（如 U-Net 等）在影像分割任務上具有極高的表現，達到更高的準確率

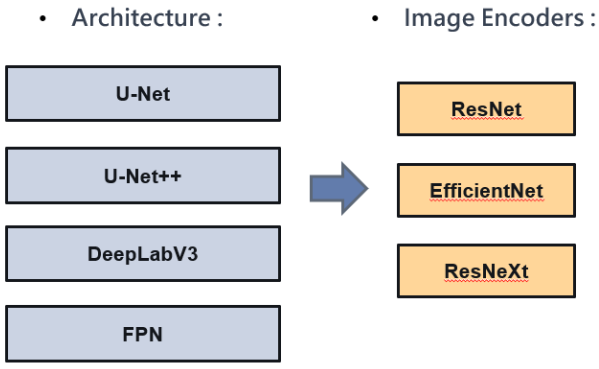
(3) **處理非線性與複雜場景**：水體影像可能包含非線性特徵、光照變化、反射等複雜因素，傳統方法難以捕捉這些細微變化。而深度學習模型具備強大的非線性映射能力，可以更好地處理複雜的影像場景。

接下來我們會詳細介紹 Deep Learning-based Method 並進行效能的評估。

2. Methodology :

(1) 深度模型介紹:

在這次實作中，我們使用了四種 Architecture 和三種 Image Encoders，總共 12 種的組合來針對這次的任務進行處理。

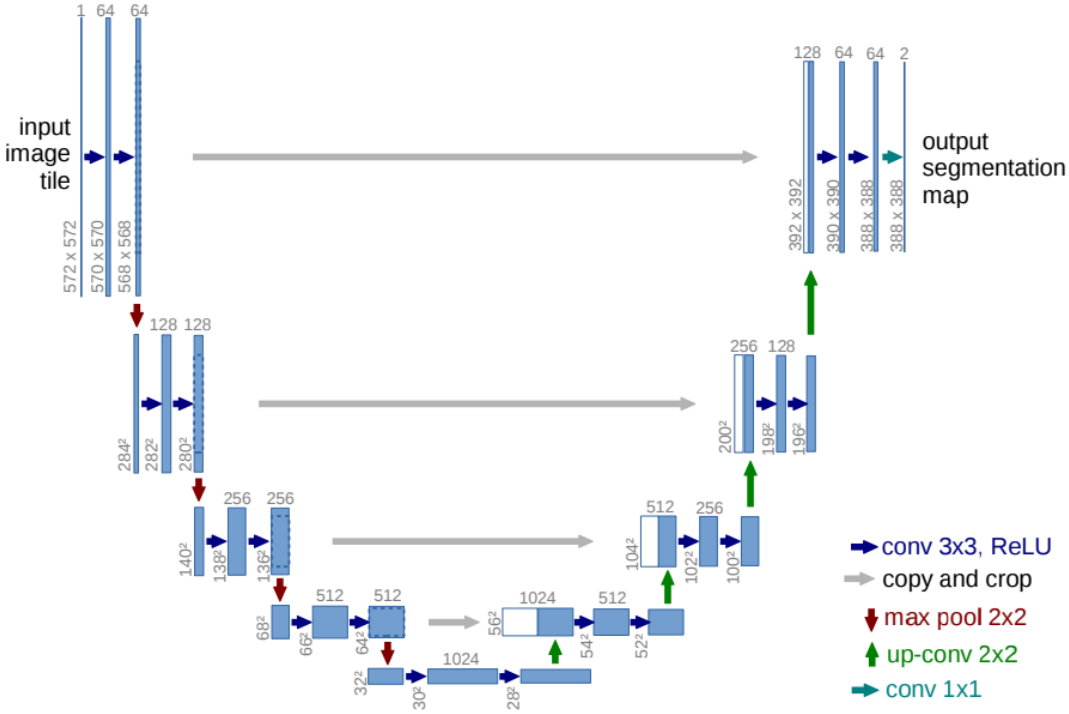


以下是圖中四種 Architecture 和三種 Image Encoders 的詳細介紹：

Architecture:

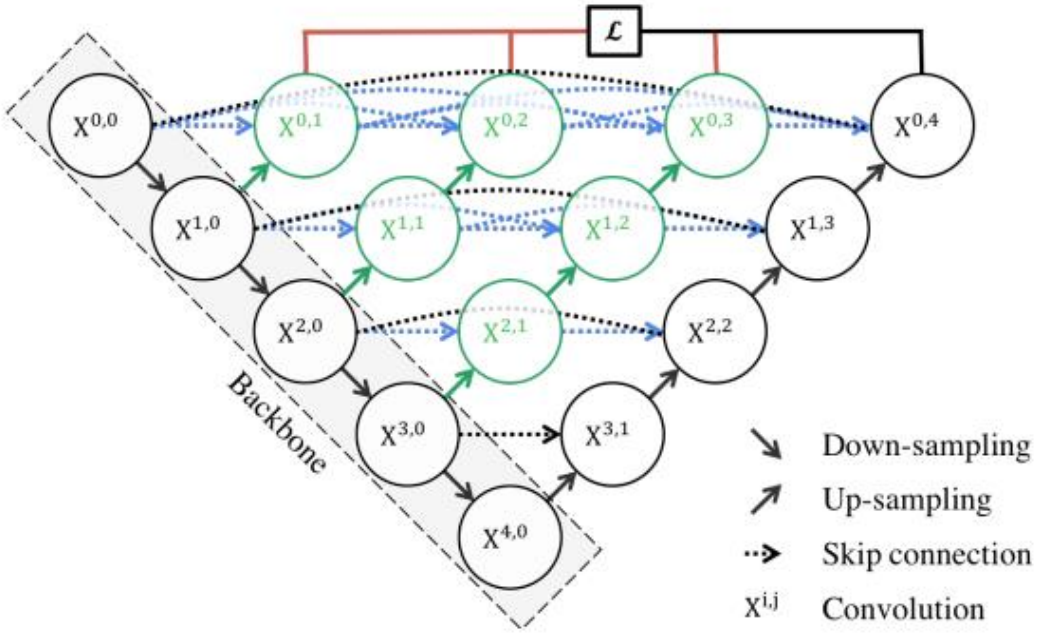
1. U-Net：

- 一種專為影像分割設計的深度學習架構
- 使用跳躍連接（Skip Connections），保留低階特徵以提高分割精度



2. U-Net++：

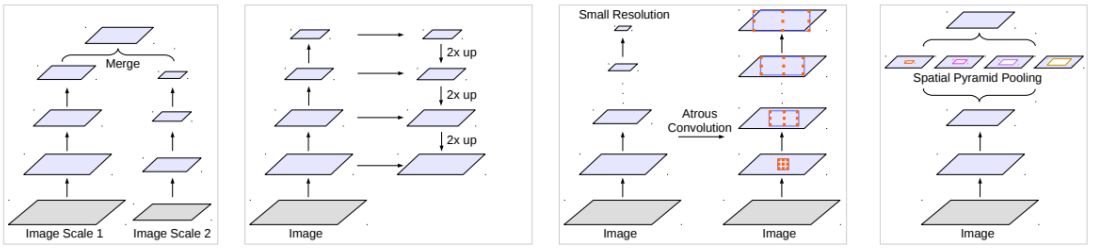
- U-Net 的改進版本，在跳躍連接中引入了密集連接，形成更深層的網路。
- 提高了分割結果的細緻度和準確率，特別適合高解析度影像分割



3. DeepLabV3：

- 使用空洞卷積（Atrous Convolution）捕捉不同尺度的語意資訊。
- 引入 ASPP（空間金字塔池化）增強網路對多尺度特徵的理解，常用於處理

複雜場景中的影像分割。



(a) Image Pyramid (b) Encoder-Decoder (c) Deeper w. Atrous Convolution (d) Spatial Pyramid Pooling

4. FPN (Feature Pyramid Network)：

- 基於金字塔結構的設計，從不同層級的特徵提取資訊。
- 提供強大的多尺度特徵表示能力，有助於檢測不同大小的物體或區域。

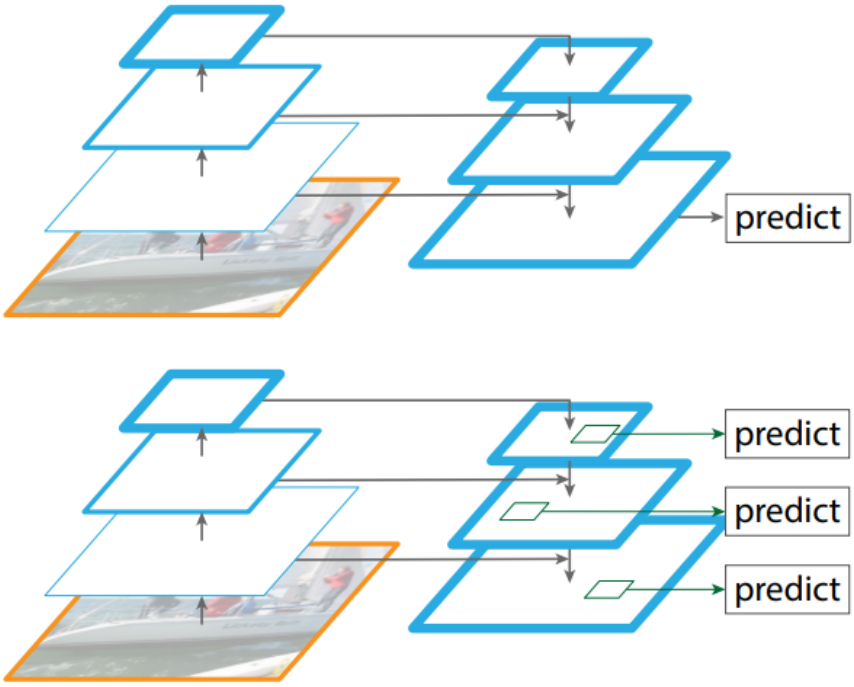


Image Encoder:

1. ResNet：

- 引入殘差連接（Residual Connections）解決梯度消失問題。
- 支持訓練非常深的網路（如 ResNet-50、ResNet-101 等）。

2. EfficientNet：

- 透過模型縮放策略（Compound Scaling）在網路深度、寬度和解析度間達到最佳平衡。
- 相較於 ResNet，同樣效能下所需的參數量更少。

3. ResNeXt：

- ResNet 的改進版本，採用分組卷積（Grouped Convolution）提高效能。
- 提供更強大的特徵提取能力，且能在參數數量與效能間取得平衡。

(2) 訓練流程:

接著我們要簡單介紹這次 Project 的程式碼，可能會和最終繳交的程式碼略有不同，但大致上功能相似。

1. 掛載 Google Drive 和安裝必要套件。

```
from google.colab import drive
drive.mount('/content/drive')

import os
os.chdir('/content/drive/MyDrive/water_segmentation_ver3')
os.listdir()
```

```
!pip install pytorch-lightning
!pip install segmentation-models-pytorch
```

2. 引入必要模組：其中 UNETModule 是從自定義模型檔案中匯入的模組，可以選擇各種不同的 Architecture 和 Encoder 組合；而 PyTorch 和 PyTorch Lightning 用於模型訓練。最後設置 random seed 保證訓練結果唯一。

```
from models.unet import UNETModule
import torch
import numpy as np
import pytorch_lightning as pl
from torch.utils.data import DataLoader
import cv2

import gc
torch.cuda.empty_cache()
gc.collect()

from datasets.water_bodies_dataset import SimpleWaterBodiesDataset
from datasets.water_bodies_dataset import PredictionWaterBodiesDataset
import albumentations as A
from albumentations.pytorch import ToTensorV2
import os
import random
from pytorch_lightning import seed_everything

myseed = 6666  # set a random seed for reproducibility
seed_everything(myseed, workers=True)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(myseed)
random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)
```

3. 設定 Dataset 與 DataLoader：A.Compose([ToTensorV2()]) 將影像轉換為 PyTorch 所需的 Tensor 格式。接著將訓練和驗證資料進行預處理並根據 batch size，每次將 4 張圖片載入模型進行訓練。因為這次訓練資料實在太少，所以我們最後決定拿所有 80 張圖片下去訓練，在這裡代表我們的訓練集和驗證集其實是一樣的，最後取 Train IOU(或稱為 Valid IOU)結果最好的模型。

```
transform = A.Compose([
    ToTensorV2()
])

root = "dataset/"
train_dataset = SimpleWaterBodiesDataset(root, mode="all", transform=transform)
val_dataset = SimpleWaterBodiesDataset(root, mode="all", transform=transform)

print("Train dataset length:", len(train_dataset))
print("Val dataset length:", len(val_dataset))

n_cpu = os.cpu_count()
train_dataloader = DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers=n_cpu)
valid_dataloader = DataLoader(val_dataset, batch_size=4, shuffle=False, num_workers=n_cpu)
```

4. 訓練過程：UNETModule 的參數 model 和 encoder 可以修改 Encoder 和 Architecture，encoder weights 設置為 None 代表從頭訓練，最後使用的損失函數為 crossentropy 作為分類的 Loss，並訓練 100 個 Epoch，中間進行反向傳播、梯度更新。

```
accelerator = "gpu" if torch.cuda.is_available() else "cpu"

unet_module = UNETModule(
    model="FPN",
    encoder="efficientnet-b4",
    encoder_weights=None,
    loss_fn="crossentropy",
)

unet_trainer = pl.Trainer(
    max_epochs=100,
    accelerator=accelerator,
    devices=1,
    log_every_n_steps=5,
    enable_checkpointing=False,
    deterministic=True,
)

unet_trainer.fit(unet_module, train_dataloader, valid_dataloader)
```

```
predict_dataset = SimpleWaterBodiesDataset(root, mode="all", transform=transform)
test_dataloader = DataLoader(predict_dataset, batch_size=20, shuffle=False, num_workers=n_cpu)
unet_trainer.validate(model=unet_module, dataloaders=test_dataloader)
```

5. 資料分析和視覺化：TensorBoard 是一個用於模型訓練過程可視化的工具，可以查看損失、準確率等指標的變化情況；lightning_logs/ 是 PyTorch Lightning 預設的日誌存儲目錄；matplotlib.pyplot 作為視覺化工具來顯示影像和預測結果。

```
%load_ext tensorboard
%tensorboard --logdir lightning_logs/
```

6. 輸出驗證集結果：將圖片影像、Ground Truth Mask 和 Prediction Mask 做比較。

```
import matplotlib.pyplot as plt

batch = next(iter(test_dataloader))
with torch.no_grad():
    unet_module.eval()
    logits = unet_module(batch["image"].float())
preds = (logits.sigmoid() > .5).float()
```

```
for image, gt_mask, pr_mask in zip(batch["image"], batch["mask"], preds):
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(image.detach().cpu().numpy().transpose(1, 2, 0))  # convert CHW -> HWC
    plt.title("Image")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(gt_mask.squeeze().detach().cpu().numpy()) # just squeeze classes dim, because we ha
    plt.title("Ground truth")
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(pr_mask.squeeze().detach().cpu().numpy()) # just squeeze classes dim, because we ha
    plt.title("Prediction")
    plt.axis("off")

    plt.show()
```

7. 輸出測試集結果：測試所有測試集的影像並將結果儲存，測試集是最終評分用的 10~20 張影像。

```python
for i in range(1, 21):
    image_test, image_size = getimage('dataset2/image/'+str(i)+'.jpg')
    with torch.no_grad():
        unet_module.eval()
        logits = unet_module(image_test.unsqueeze(0).float())
    preds = (logits.sigmoid() > .5).float()

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image_test.detach().cpu().numpy().transpose(1, 2, 0))  # convert CHW -> HWC
    plt.title("Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(preds.squeeze().detach().cpu().numpy()) # just squeeze classes dim, because we have
    plt.title("Prediction")
    plt.axis("off")

    plt.show()


    preds = preds.squeeze().detach().cpu().numpy()
    preds = preds*255
    preds = Image.fromarray(preds.astype(np.uint8))
    preds = preds.resize(image_size, Image.NEAREST)
    preds.save('dataset2/output/'+str(i)+'.png')
```

3. Result :

(1) 各個 Architecture – Encoder 組合輸出的 prediction 和 ground truth 之間的 IOU Score，分數包括訓練集的 IOU（或驗證集的 IOU，也就是全部 80 張影像)以及測試集的 IOU:

Architecture – Encoder	IOU Score	Params(M)
U-Net – Resnet	**0.9792/0.4549**	**14.3**
U-Net – Efficientnet	**0.9583/0.4783**	**20.2**
U-Net – ResNext	**0.9535/0.3483**	**32.0**
U-Net++ – Resnet	**0.9844/0.3956**	**16.0**
U-Net++ – Efficientnet	**0.9566/0.4862**	**20.8**
U-Net++ – ResNext	**0.9511/0.4391**	**48.5**
DeepLabV3 – Resnet	**0.9776/0.4527**	**15.9**
DeepLabV3 – Efficientnet	**0.9665/0.4407**	**21.8**
DeepLabV3 – ResNext	**0.9706/0.3628**	**39.1**

FPN – Resnet	**0.9791/0.4227**	**13.0**
FPN – Efficientnet	**0.9459/0.4295**	**19.4**
FPN – ResNext	**0.9627/0.4038**	**25.6**

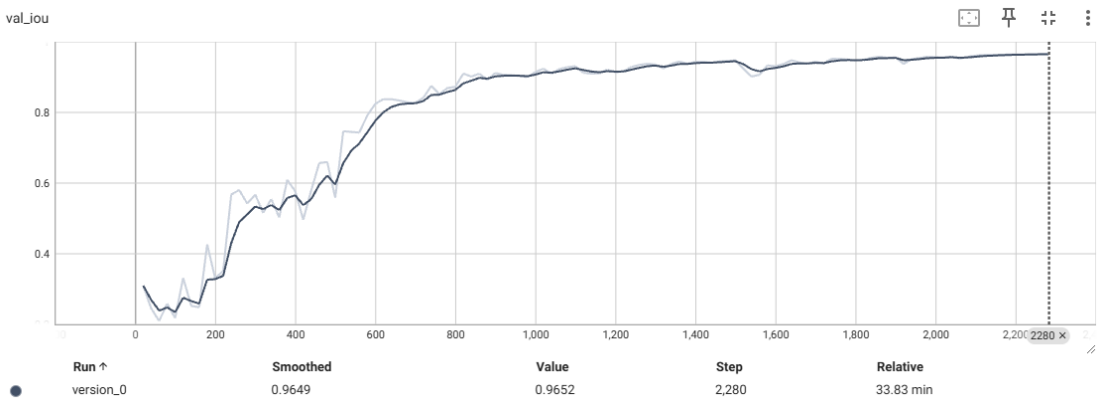
以下是平均每一種 Encoder 的 IOU Score 來做比較:

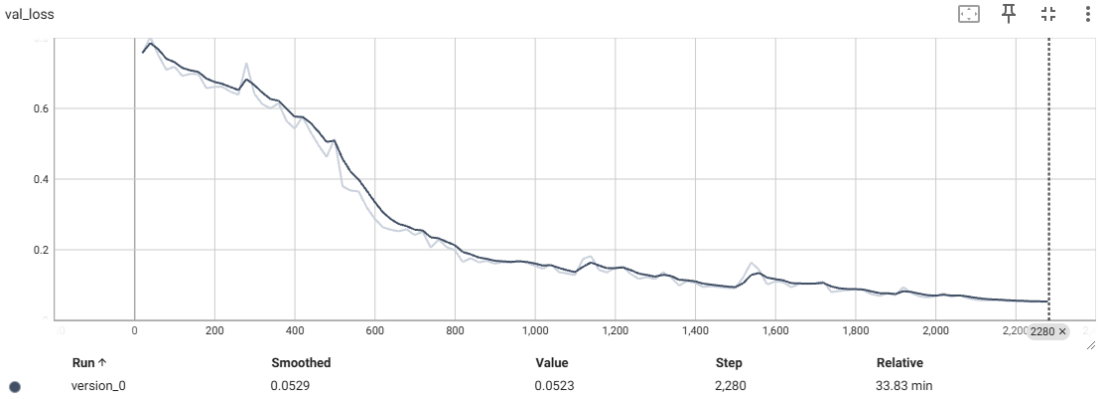
Encoder	Average IOU Score
Resnet	**0.9801/0.4314**
Efficientnet	**0.9568/0.4586**
ResNext	**0.9594/0.3885**

以下是平均每一種 Architecture 的 IOU Score 來做比較:

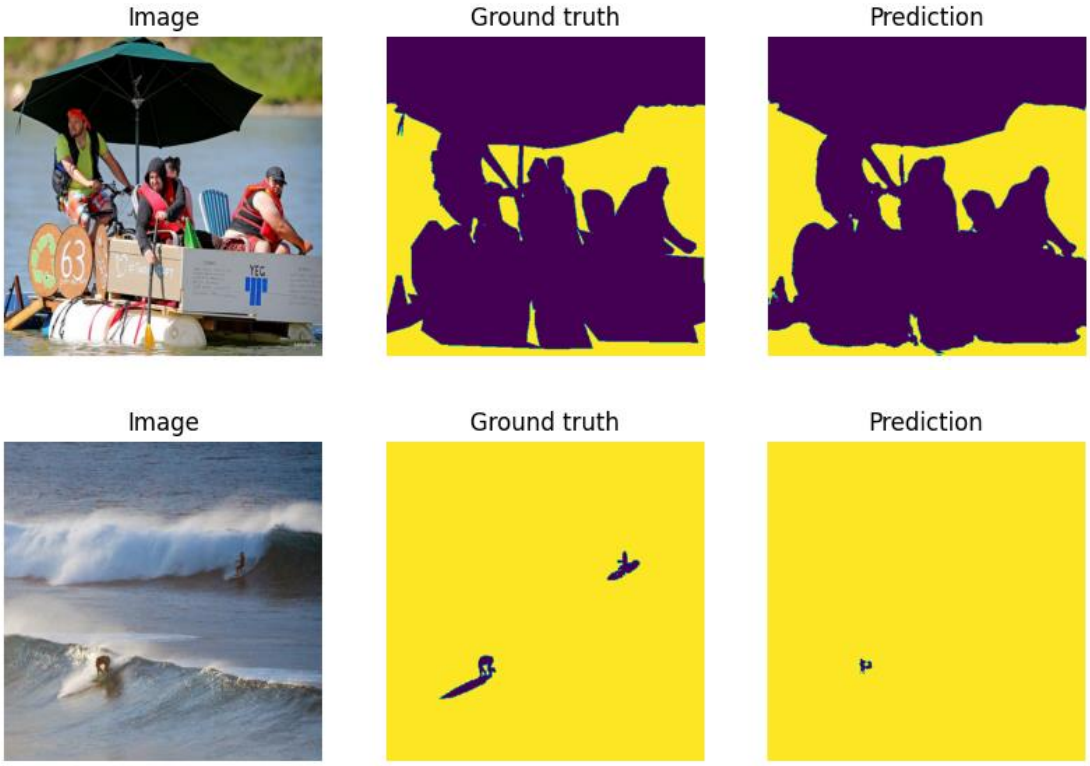
Architecture	Average IOU Score
U-Net	**0.9636/0.4271**
U-Net++	**0.9641/0.4403**
DeepLabV3	**0.9715/0.4187**
FPN	**0.9625/0.4186**

(2) TensorBoard 資料分析和視覺化(U-Net++ – Efficientnet)，呈現出模型在訓練時 IOU 變化和 Loss 變化的情況，可以看出大概在訓練到 Epoch 50 左右模型就變穩定:

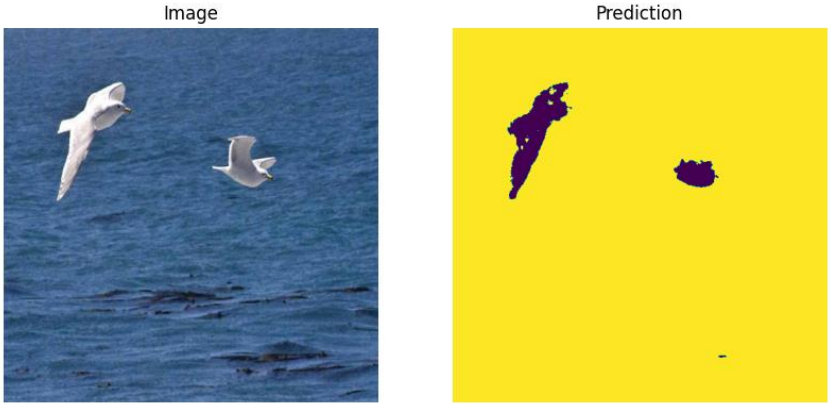


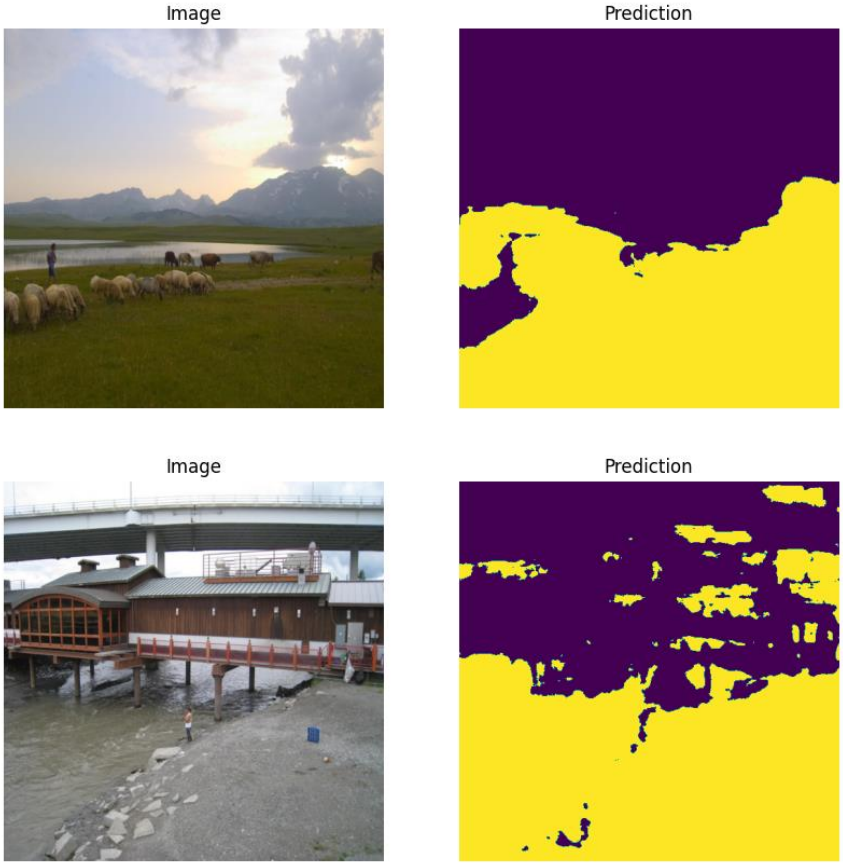


(3) 訓練集(或是驗證集)輸出結果:



(4) 測試集輸出結果:





(5) 結論:

1. 相較於傳統影像分割方法，深度學習方法更能擷取影像中重要的資訊來分割更複雜的場景。在訓練集(或驗證集)中 IOU 平均可以來到 0.9 以上，比原本傳統方法的 0.2~0.3 還要高出許多。可以看到在(3)之中針對浪花等複雜的場景使用深度學習模型可以分割得更加精確，相比第一頁使用傳統影像分割方法還要來的好。

2. 我們整理完各種 Architecture 和 Encoder 組合的 IOU Score 以及所有 Architecture 和 Encoder 的平均 IOU Score 在(1)的表格中。我們除了查看第一個表格最好的 IOU Score，還有計算各個 Architecture 和 Encoder 的平均 IOU 再決定我們最終選擇的模型。

3. 可以發現在 Encoder 的部份雖然平均下來 Resnet 在訓練集的 IOU Score 較高，但在測試集中反而是 Efficientnet 比較高，再加上 Efficientnet 在訓練集的 IOU Score 也不低，所以我們最終選擇 Efficientnet 當作 Encoder。從文獻中可以知道 Efficientnet 透過模型縮放策略（Compound Scaling）和網絡結構搜尋（Neural Architecture Search）的方法能夠找到更好的特徵資訊，是非常強大的特徵提取器，所以最後選擇 Efficientnet 當作 Encoder 是比較合理的

選擇。

4. 而在 Architecture 的部分 U-Net++也有相同的情況，訓練集 IOU Score 雖然沒有最好,但在測試集中卻是最高的。我們可以看到前兩名被 UNet 及 UNet++佔據，可見這些較早期但經典的模型依舊在某些時刻具有較強的功效。接著 UNet++跟第二名的 UNet 相比較,UNet++ 透過增加多層次的跳躍連接和增強的特徵融合機制，改進了 UNet 在影像分割中的表現，特別是在細節捕捉和多尺度學習上有顯著優勢，因此有較好的分數。其次也可以從(1)知道 U-Net++ - Efficientnet 的 IOU Score 是位居第一的模型組合，所以最後我們選擇 U-Net++ - Efficientnet 這個模型組合來繳交作業。

5. 最終測試集 IOU 平均下來只有 0.4 左右，大概只有勉強高過 Baseline 的標準,可能是因為我們的訓練集(或驗證集)和測試集的分布或是圖片的類型不一致所以測試結果沒有很好,例如(4)的範例中第一個圖片分割結果算是還不錯,但是在第二和第三張圖片遇到顏色或是陰影的干擾導致模型誤以為草原或是泥沙就是水域,所以若是能夠針對這類的資料做一些前處理或許就能有更好的結果。或著有可能只是單純訓練集太少讓模型有 Overfitting 的狀況，雖然針對資料少的部分有嘗試做資料擴增，例如 rotation, scaling, flipping, cropping 等,但效果不理想所以作罷,後來想到可以參考前面的狀況調整圖片明暗或是改變水域的顏色來做資料擴增或許才是較好的解決辦法。

6. 此外這次模型訓練我們採用 Pytorch Lightning，他在保有 PyTorch 的原生功能之下簡化了程式碼結構，讓我們的程式更加整潔和易於維護，且能夠提升運行效率。