Deep learning lab2 姓名:陳品翰 學號:313552042

1. Implementation Details

(1) train.py: 在這個 train.py 裡面我會先依照 oxford_pet.py 裡面的前處理方式先把 data 前處理完畢(前處理的方式第 2.有詳細描述)，之後會用 load_dataset 這個函式把資料 load 進來，之後指定要用 Unet 或是 Resnet34 神經網路，指定好以後，用 AdamW 這個 optimizer，並用 BCEloss 為 loss function，之後便開始做訓練，另外，我這裡面可以調整是否要把(1-dice score)也考慮進 loss function 中，可以調整他和 BCEloss 的比例，下面的報告有描述我用 0.5 的 BCEloss 加上 0.5 的(1-dice score)作為 loss function 的結果，這部分的比例是可以調整的，之後每 train 一個 epoch，會對 validation set 做一次預測，來看看目前的模型表現如何，最後把 train loss/epoch 和 validation loss/epoch 圖表存取下來，也把模型存下來。

```
def train(args):
    """ 訓練 UNet 模型 """

    # 設定設備
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # 載入數據集
    train_loader = load_dataset(args.data_path, mode="train", batch_size=args.batch_size)
    val_loader = load_dataset(args.data_path, mode="valid", batch_size=args.batch_size)
    # 初始化 UNet 模型
    #model = UNet(in_channels=3, out_channels=1).to(device)
    model = ResNet34UNet(in_channels=3, out_channels=1).to(device)
    # 損失函數 & 優化器
```

可以在這邊選要 load 的模型

(2) evaluate.py:這個 code 會由 inference.py 這個檔案呼叫，evaluate 本身會把 model 轉換為 eval()模式後，對 test set 做測試，同時呼叫 utils.py 裡面計算 dice score 的函式來印出 dice score。

(3) inference.py:會用 load_dataset 這個函式把 test set load 進來，之後呼叫 evaluate 來計算 Model 在 test set 上面的表現。

```
def load_model(model_path, device):
    """ 加載已訓練的 UNet 模型 """
    model = UNet(in_channels=3, out_channels=1).to(device)
    #model = ResNet34UNet(in_channels=3, out_channels=1).to(device)
    model.load_state_dict(torch.load(model_path, map_location=device))
    model.eval()  # 設置為推論模式
    return model
```

這邊可以調整要 Load 哪個模型

(4).Unet.py: Unet 的實作包含 Encoder、bottleneck、Decoder 這三個部分，其

中 Encoder 是有 4 個 block 組成，每個 block 會做兩次的 conv 跟一次 maxpooling，並且把做完的結果存起來，之後經過一個 bottleneck(兩次 conv)後，會接上 Decoder，這個 decoder 會把 encoder 做完時的結果和目前做完 ConvTranspose2d 的結果連結在一起(encoder 的結果傳去給 decoder 的這個動作就是 skip connection)，傳給下一個 layer，而 decoder 要連接的 encoder 結果在 U 字形中是對稱的，也就是第一個 decoder 和最後一個 encoder 的結果作連結，以此類推，即形成一個 Unet，而 encoder 的 block，架構為:

```
def double_conv(self, in_channels, out_channels):
    """ 兩層 3x3 Conv + ReLU """
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
    )
```

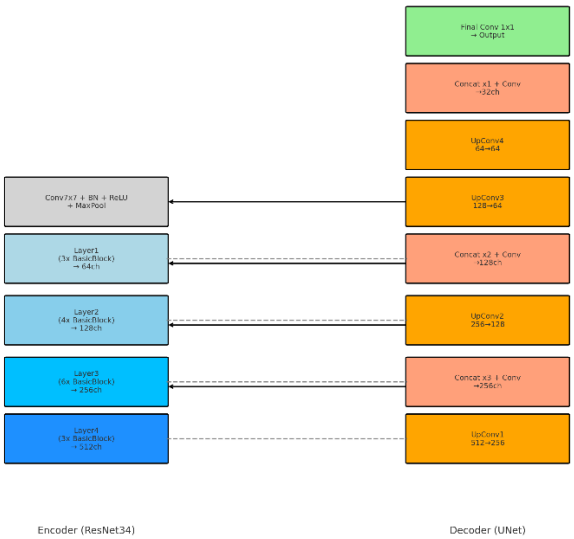
經過我的實驗後，我發現需要加上 Batch Normalization 這個步驟，才可以有優秀的表現。

(5).Resnet34Unet.py:這個神經網路主要是用 Resnet34 當成 Encoder，和 Unet 的 decoder 串接起來的神經網路，並且和 Unet 一樣有 skip connection，Resnet34 的 encoder 和 Unet 的 encoder 不一樣的是，他有 residual 的部分，也就是把 input 和做完的 output 做相加的動作，這在實作上為:

```
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        if self.downsample is not None:
            identity = self.downsample(x)

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out += identity
        out = self.relu(out)

        return out
```

可以看到 forward 裡面，會把做完的結果和 identity 相加，另外 Restnet34 的 encoder 也比 Unet 的 encoder 複雜得多，他在每一個 layer 裡面會有多個 basic block，從前到後 layer 的 basic block 的個數為 3、4、6、3，依照這個架構建完 encoder 後，接著就是 Unet 的 decoder，這邊一樣有 skip connection 的架構，也需要注意圖片的大小要和 decoder 的相同，其他的部分和上面介紹過的 Unet decoder 都非常相似。

附圖是 rest34Unet 的架構圖，虛線部分是 skip connection，另外實線箭頭從 decoder 到 encoder 的意思是要去 encoder 拿取資料來做 concat



Unet 裡面的 skip connection 的目的為，因為在做 conv 跟 pooling 時，會犧牲掉很多的細節資訊，所以需要靠 encoder 那邊那些較高解析度的圖片，來幫助還原細節。

Resnet34 的 residual 部分則是可以讓學習更簡單，因為可以從學習整個 output 變成改學習 input 跟 output 的差值。

2. Data Preprocessing

```python
# **定義影像增強**
self.augmentation = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),  # 50% 機率水平翻轉
    transforms.RandomRotation(degrees=20),  # 隨機旋轉 ±20度
]) if mode == "train" else None

# **定義影像轉換**
self.image_transform = transforms.Compose([
    transforms.Resize((256, 256)),  # 影像縮放
    transforms.ToTensor(),  # 轉換為 Tensor
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # 標準化影像
])

# **定義 Mask 轉換**
self.mask_transform = transforms.Compose([
    transforms.Resize((256, 256), interpolation=Image.NEAREST),  # 確保不影響標籤
    transforms.ToTensor()  # 轉換為 Tensor
])
```

```python
# 進行 augmentation（確保 image 和 mask 用相同的隨機參數）
seed = torch.randint(0, 1000000, (1,)).item()  # 生成固定 seed
torch.manual_seed(seed)  # 設定 seed，確保 image & mask 變換一致
image = self.augmentation(image)

torch.manual_seed(seed)  # 再次設定相同的 seed，確保 mask 變換一致
```

當 training set 傳進去 model 以前，我會對圖片做水平翻轉跟隨機旋轉等等動作，之所以沒有做上下翻轉的原因是因為不合理，在正常的資料集中應該不會有上下顛倒的動物，而 data argumentation 的重點即是要想辦法增加資料量，但是增加的資料必須是合理的，否則會破換訓練，另外因為我的 training set 有做 transform，我相對應的 mask 也要做一樣的動作，這樣才可以學習到正確的答案，我曾經有只轉 training 圖片的情況，這樣做的結果 dice score 只可以達到 0.81 左右，當 mask 跟 image 做一樣的 transform 時，因為可以學習到正確的答案，dice score 可到達 0.91 左右。

3. Analyze the experiment results

當我一開始沒有做 data argumentation 時，我的 Unet dice score 只有 0.89 附

近，幾乎不可能可以到達 0.90 這個門檻，所以我後來去做了 transform，並且確認我的 image 跟 mask 都有一起做轉換後，我的 dice score 就可以到達 0.91 附近了，用的 batch size 為 16，epoch 為 50，learning rate 為 0.001，如果 batch size 改為 8，則在 epoch 20 初就過了 0.9，但最後的結果並沒有比較高，所以我最後採用了 batch size 16，因為這樣可以增快訓練的速度。

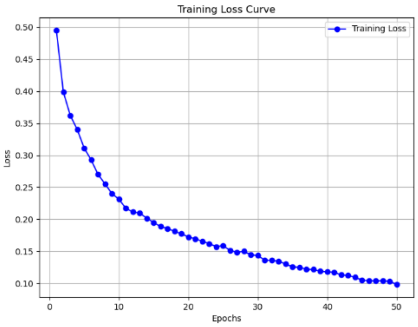
我一開始使用的 loss function 為 BCEloss，可以觀察到最後 train 出來的結果已經很不錯了，之後我嘗試把 1 – dice score 也加入到 loss 裡面，算式為 0.5*(1-dice score) + 0.5 * BCEloss 後，可以看到結果也很不錯

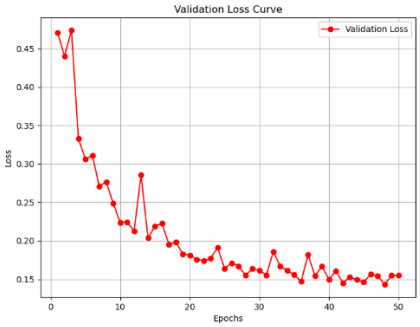
附圖為 Unet BECloss 的結果: 上圖為 training/validation 下圖為 testing

```
Epoch [48/50], Loss: 0.1042
Average Dice Score: 0.9217, Pixel Accuracy: 0.9452
Validation Loss: 0.1434, Dice Score: 0.9217, Pixel Accuracy: 0.9452
Epoch [49/50], Loss: 0.1028
Average Dice Score: 0.9199, Pixel Accuracy: 0.9423
Validation Loss: 0.1548, Dice Score: 0.9199, Pixel Accuracy: 0.9423
Epoch [50/50], Loss: 0.0984
Average Dice Score: 0.9215, Pixel Accuracy: 0.9439
Validation Loss: 0.1548, Dice Score: 0.9215, Pixel Accuracy: 0.9439
Model training completed and saved!
Loss curve saved as 'loss_curve.png'
Validation Loss curve saved as 'validation_loss_curve.png'
Dice Score curve saved as 'dice score curve.png'
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/unet.pth --data_path ../dataset/ox
ford-iiit-pet --batch_size 16
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9252, Pixel Accuracy: 0.9445
Test dice_score: 0.9252, Pixel Accuracy: 0.9445
```

Unet BCEloss training loss/epoch validation loss/epoch batch size:16 learning_rate:0.001

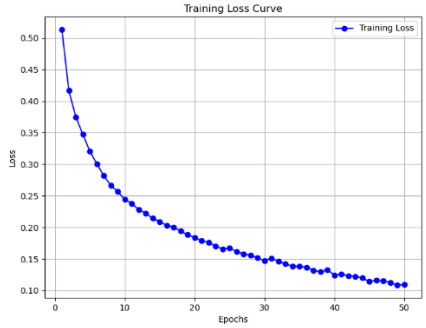


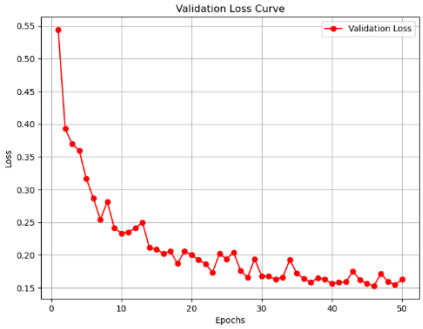


Batch size:8 learning_rate:0.001

```
Epoch [47/50], Loss: 0.1155
Average Dice Score: 0.9081, Pixel Accuracy: 0.9366
Validation Loss: 0.1719, Dice Score: 0.9081, Pixel Accuracy: 0.9366
Epoch [48/50], Loss: 0.1128
Average Dice Score: 0.9181, Pixel Accuracy: 0.9423
Validation Loss: 0.1591, Dice Score: 0.9181, Pixel Accuracy: 0.9423
Epoch [49/50], Loss: 0.1086
Average Dice Score: 0.9183, Pixel Accuracy: 0.9414
Validation Loss: 0.1551, Dice Score: 0.9183, Pixel Accuracy: 0.9414
Epoch [50/50], Loss: 0.1092
Average Dice Score: 0.9127, Pixel Accuracy: 0.9369
Validation Loss: 0.1628, Dice Score: 0.9127, Pixel Accuracy: 0.9369
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/unet.pth --data_path ../dataset/ox
ford-iiit-pet --batch_size 8
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9166, Pixel Accuracy: 0.9385
Test dice_score: 0.9166, Pixel Accuracy: 0.9385
```

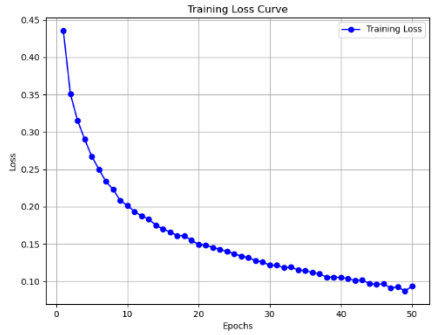


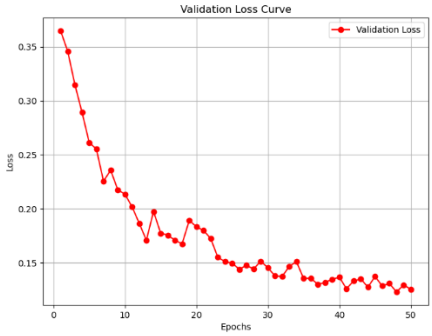


Unet 0.5* BCEloss + 0.5 * (1 – dice score) batch size:16 learning_rate:0.001

```
Epoch [47/50], Loss: 0.0909
Average Dice Score: 0.9202, Pixel Accuracy: 0.9422
Validation Loss: 0.1310, Dice Score: 0.9202, Pixel Accuracy: 0.9422
Epoch [48/50], Loss: 0.0925
Average Dice Score: 0.9246, Pixel Accuracy: 0.9451
Validation Loss: 0.1232, Dice Score: 0.9246, Pixel Accuracy: 0.9451
Epoch [49/50], Loss: 0.0869
Average Dice Score: 0.9211, Pixel Accuracy: 0.9433
Validation Loss: 0.1296, Dice Score: 0.9211, Pixel Accuracy: 0.9433
Epoch [50/50], Loss: 0.0934
Average Dice Score: 0.9211, Pixel Accuracy: 0.9425
Validation Loss: 0.1256, Dice Score: 0.9211, Pixel Accuracy: 0.9425
Model training completed and saved!
Loss curve saved as 'loss_curve.png'
Validation Loss curve saved as 'validation_loss_curve.png'
Dice Score curve saved as 'dice_score_curve.png'
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/unet_use_dicesocre_loss.pth --data
 _path ../dataset/oxford-iiit-pet --batch_size 16
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9252, Pixel Accuracy: 0.9438
Test dice score: 0.9252, Pixel Accuracy: 0.9438
```

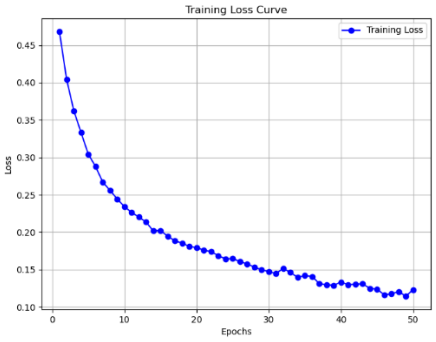


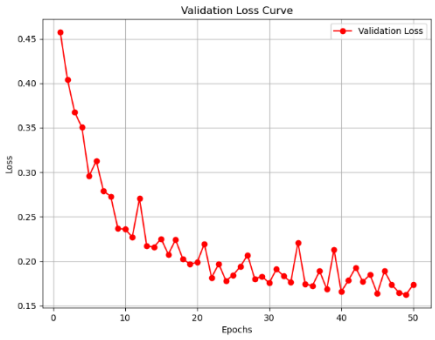


Resnet34Unet BCEloss batch_size = 16 learnin_rate = 0.001

```
Epoch [48/50], Loss: 0.1205
Average Dice Score: 0.9093, Pixel Accuracy: 0.9377
Validation Loss: 0.1647, Dice Score: 0.9093, Pixel Accuracy: 0.9377
Epoch [49/50], Loss: 0.1145
Average Dice Score: 0.9105, Pixel Accuracy: 0.9364
Validation Loss: 0.1624, Dice Score: 0.9105, Pixel Accuracy: 0.9364
Epoch [50/50], Loss: 0.1233
Average Dice Score: 0.9114, Pixel Accuracy: 0.9372
Validation Loss: 0.1739, Dice Score: 0.9114, Pixel Accuracy: 0.9372
Model training completed and saved!
Loss curve saved as 'loss_curve.png'
Validation Loss curve saved as 'validation_loss_curve.png'
Dice Score curve saved as 'dice_score_curve.png'
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/resnet34_unet.pth --data_path ../d
ataset/oxford-iiit-pet --batch_size 16
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9196, Pixel Accuracy: 0.9399
Test dice score: 0.9196, Pixel Accuracy: 0.9399
```

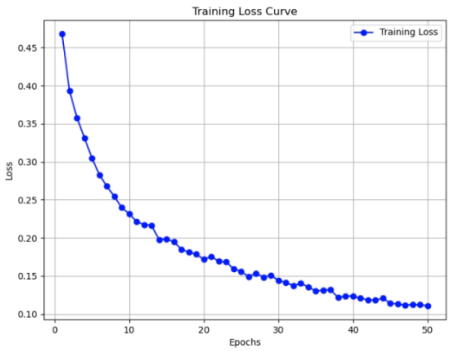


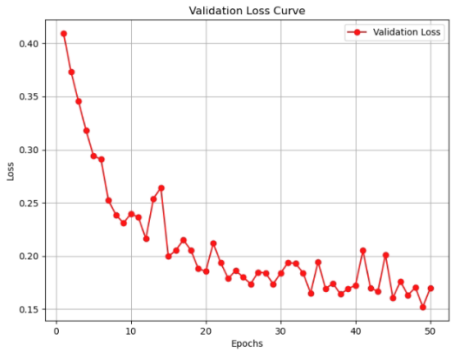


batch_size = 8 learnin_rate = 0.001

```
Average Dice Score: 0.9098, Pixel Accuracy: 0.9377
Validation Loss: 0.1704, Dice Score: 0.9098, Pixel Accuracy: 0.9377
Epoch [49/50], Loss: 0.1122
Average Dice Score: 0.9203, Pixel Accuracy: 0.9428
Validation Loss: 0.1521, Dice Score: 0.9203, Pixel Accuracy: 0.9428
Epoch [50/50], Loss: 0.1109
Average Dice Score: 0.9161, Pixel Accuracy: 0.9385
Validation Loss: 0.1701, Dice Score: 0.9161, Pixel Accuracy: 0.9385
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/resnet34unet_8.pth --data_path ../
dataset/oxford-iiit-pet --batch_size 8
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9216, Pixel Accuracy: 0.9407
Test dice score: 0.9216, Pixel Accuracy: 0.9407
```



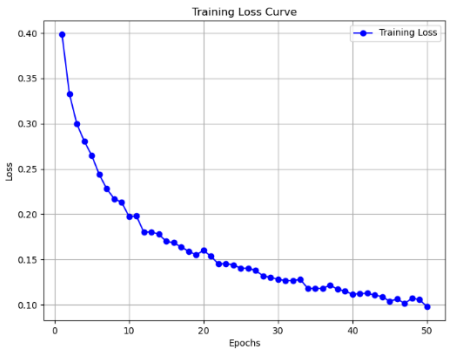


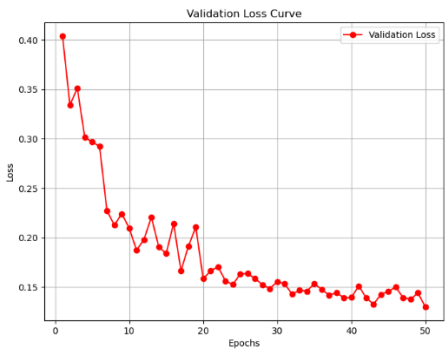
Resnet34Unet 0.5* BCEloss + 0.5 * (1 – dice score) batch size:16

learning_rate:0.001

```
Validation Loss: 0.1377, Dice Score: 0.9157, Pixel Accuracy: 0.9386
Epoch [49/50], Loss: 0.1056
Average Dice Score: 0.9158, Pixel Accuracy: 0.9393
Validation Loss: 0.1439, Dice Score: 0.9158, Pixel Accuracy: 0.9393
Epoch [50/50], Loss: 0.0980
Average Dice Score: 0.9193, Pixel Accuracy: 0.9433
Validation Loss: 0.1299, Dice Score: 0.9193, Pixel Accuracy: 0.9433
Model training completed and saved!
Loss curve saved as 'loss_curve.png'
Validation Loss curve saved as 'validation_loss_curve.png'
Dice Score curve saved as 'dice_score_curve.png'
```

```
(dl) Roy@wnlab-4080:~/nas/home/deep_learning/Lab2_Binary_Semantic_Segmentation_2025/src$ python inference.py --model ../saved_models/restnet34unet_use_dicesocre_loss.p
th --data_path ../dataset/oxford-iiit-pet --batch_size 16
Using device: cuda
Evaluating on Test Set...
Average Dice Score: 0.9195, Pixel Accuracy: 0.9401
Test dice score: 0.9195, Pixel Accuracy: 0.9401
```





可以觀察到不同的 batch size，不一樣計算 loss 的方法，得到最後的結果是差不多的，由此可知這應該就是這兩個模型的極限了。

4. Execution steps

執行 train.py 的 command 為:

python train.py --data_path ../dataset/oxford-iiit-pet --epochs 50 --batch_size 16 --learning-rate 0. 001

data path:就是放置資料集的路徑

batch size: train 時的 batch size 可以在 command line 設定

learning rate: learning rate 也可以在 command line 設定

要 load 哪個 model 在第 1.那邊有描述

執行 inference.py 的 command 為:

python inference.py --model ../saved_models/resnet34unet_8.pth --data_path ../dataset/oxford-iiit-pet --batch_size 8

--model:後面是 model 的路徑

--data_path:後面是 data set 的 path

--batch size:是 test set 在測試時的 batch size

5. Discussion:

SegNet 或許是一個適合的模型架構，雖然同樣是 encoder-decoder 的架構，但是他在 decoder 利用最大池化索引來進行反向傳播，有助於保留邊界資訊，對於我們現在處理的這個資料集，SegNet 在邊界處理上可能會有一些優勢。

研究方向:可以朝無監督學習與半監督學習的方向考慮，因為 dataset 的標註資料相對有限，所以無監督或半監督的方式可以在標註不足的情況下提高分割的效果。