Lab2 Report

Deep Learning

Binary_Semantic_Segmentation

姓名：陳科融

學號：314551010

July 10, 2025

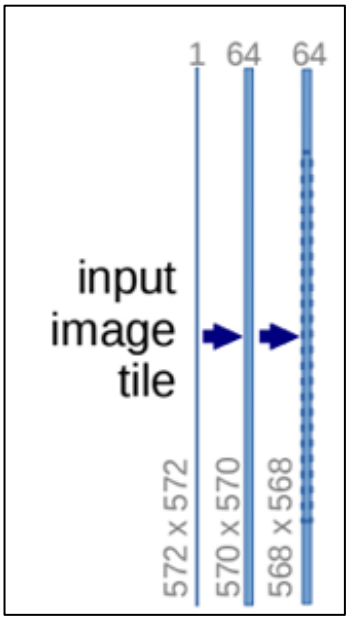
1. Implementation Details

1.1 Detail of model

1.1.1 UNet

DoubleConv(Conv2d+BatchNorm2d+Relu)

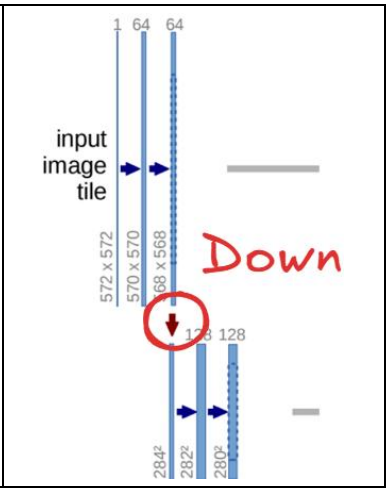
```
class DoubleConv(nn.Module):
    """
    一個包含兩個卷積層的區塊: (卷積 -> Batch Normalization -> ReLU)*2
    Conv->BatchNorm->ReLU->Conv->BatchNorm->ReLU
    """
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels # 預設用 out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
    def forward(self, x):
        return self.double_conv(x)
```



程式碼 1. 這一部分的程式碼，有兩個卷積層區塊，如下 Conv->BatchNorm->ReLU->Conv->BatchNorm->ReLU

Down(下採樣區塊)

```
class Down(nn.Module):
    # 下採樣區塊: MaxPool -> DoubleConv
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),       # maxpool 2x2
            DoubleConv(in_channels, out_channels)
        )
    def forward(self, x):
        return self.maxpool_conv(x)
```

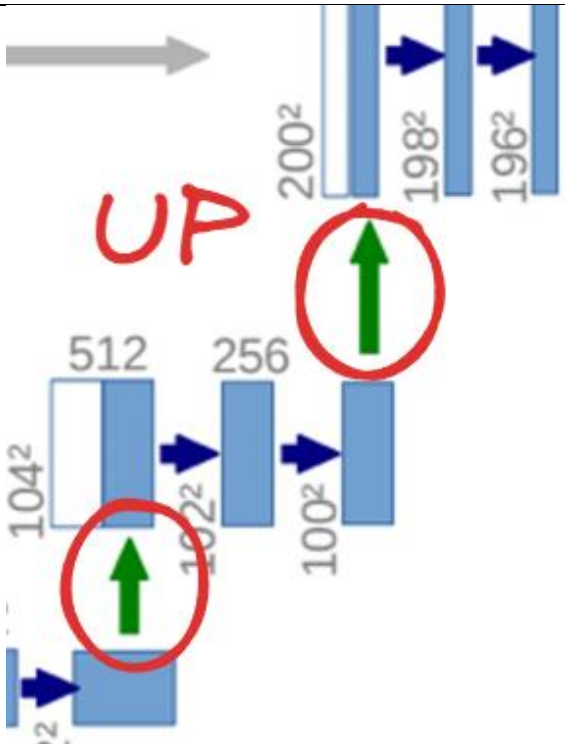


程式碼 2. 編碼器（Encoder）下採樣會需要兩次的 conv，然後做 2*2 的 maxpooling，每次 maxpooling，整個區塊的長寬會變成原先的一半。

UP(上採樣區塊)

```
class Up(nn.Module):
    # 上採樣區塊: 上採樣 -> DoubleConv
    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
        self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        diffY = x2.size()[2] - x1.size()[2] # 高度差異
        diffX = x2.size()[3] - x1.size()[3] # 寬度差異
        # F.pad(input, [左, 右, 上, 下])
        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
# 將 skip connection 的特徵圖和上採樣後的特徵圖串接
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```



程式碼 3. 解碼器（Decoder）的一部分，目的是將低解析度的特徵圖還原成高解析度，同時融合編碼器中對應層的特徵（skip connection），會將編碼器特徵圖「copy and crop」（複製並裁切）以對齊尺寸，再與上採樣後的特徵圖串接，讓模型兼顧位置細節與類別辨識。

UNet network

```
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        # 編碼器 (Contracting Path)
        self.input = DoubleConv(n_channels, out_channels=64)
        self.down1 = Down(in_channels=64, out_channels=128)
        self.down2 = Down(in_channels=128, out_channels=256)
        self.down3 = Down(in_channels=256, out_channels=512)
        self.down4 = Down(in_channels=512, out_channels=1024)
        # 解碼器 (Expansive Path)
        self.up1 = Up(in_channels=1024, out_channels=512)
        self.up2 = Up(in_channels=512, out_channels=256)
        self.up3 = Up(in_channels=256, out_channels=128)
        self.up4 = Up(in_channels=128, out_channels=64)
        # 輸出層
        self.output = OutConv(in_channels=64, out_channels=n_classes)
    def forward(self, x):
        # 編碼器
        x1 = self.input(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        # 解碼器 + Skip Connections
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        # 輸出
        return self.output(x)
```

程式碼 4. 編碼器（Contracting Path）負責逐步抽取影像特徵，並降低空間解析度，透過多層 DoubleConv 和 Down 模組實現特徵提取與下採樣。解碼器（Expansive Path）負責逐步恢復空間解析度，並融合對應編碼器層的特徵（skip connection），通過多層 Up 模組完成上採樣與特徵融合。輸出層 OutConv 則將最後解碼器輸出轉換成目標類別的像素分割結果。

原始論文中卷積層未使用 padding，使得每次卷積後特徵圖的空間尺寸會縮小，因此在跳接（skip connection）時，需手動裁切（crop）編碼器中的特徵圖，以對齊解碼器的特徵圖尺寸。而本實作中，每個卷積層皆使用 padding=1，在採用 3×3 卷積核的情況下，能保持輸入與輸出特徵圖的空間尺寸一致。此設計使得編碼器與解碼器間的特徵圖尺寸更容易對齊，避免了手動裁切的需求。

此外，為了處理輸入圖像尺寸非 16 的倍數導致的細微尺寸差異，本實作在解碼器中使用 F.pad 函數對上採樣後的特徵圖進行填補（padding），以調整特徵圖尺寸，使其與跳接特徵圖匹配，從而簡化了 skip connection 的融合過程。

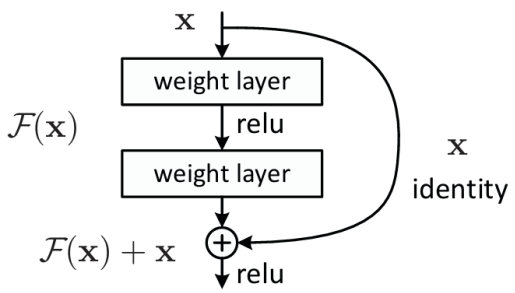
```
[Input] torch.Size([1, 3, 572, 572])
[x1] after input: torch.Size([1, 64, 572, 572])
[x2] after down1: torch.Size([1, 128, 286, 286])
[x3] after down2: torch.Size([1, 256, 143, 143])
[x4] after down3: torch.Size([1, 512, 71, 71])
[x5] after down4: torch.Size([1, 1024, 35, 35])
[x] after up1: torch.Size([1, 512, 71, 71])
[x] after up2: torch.Size([1, 256, 143, 143])
[x] after up3: torch.Size([1, 128, 286, 286])
[x] after up4: torch.Size([1, 64, 572, 572])
[Output] torch.Size([1, 1, 572, 572])
```

圖 1. U-Net 每一層特徵圖尺寸變化

1.2 ResNet34_UNet

ResNet BasicBlock

```python
class BasicBlock(nn.Module):
    expansion = 1  # 每個 block 不改變通道數
    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride
    def forward(self, x):
        identity = x # 傳入的 x，用來做 skip connection
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample is not None: # 若輸入/輸出維度不同
            identity = self.downsample(x)
        out += identity
        out = self.relu(out)
        return out
```



程式碼 5. BasicBlock 殘差區塊（Residual Block）

BasicBlock 是基礎建構模組，主要功能為學習輸入與輸出間的殘差映射，並透過 shortcut（跳接）結構 改善深層網路訓練時的梯度消失問題。

程式碼 5. ResNet BasicBlock 程式結構

ResNetEncoder

```
class ResNetEncoder(nn.Module):
    """從零開始實作的 ResNet34 編碼器"""
    def __init__(self, block, layers, n_channels=3):
        super(ResNetEncoder, self).__init__()
        self.inplanes = 64
        # 初始卷積層
        self.conv1 = nn.Conv2d(n_channels, self.inplanes, kernel_size=7, stride=2, padding=3,
bias=False)
        self.bn1 = nn.BatchNorm2d(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        # 堆疊殘差塊
        self.layer1 = self._make_layer(block, planes=64, blocks=layers[0])
        self.layer2 = self._make_layer(block, planes=128, blocks=layers[1], stride=2)
        self.layer3 = self._make_layer(block, planes=256, blocks=layers[2], stride=2)
        self.layer4 = self._make_layer(block, planes=512, blocks=layers[3], stride=2)

    def _make_layer(self, block, planes, blocks, stride=1):
        downsample = None
        # 尺寸要縮小，輸入與輸出維度相同才能 out += identity
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride),
                nn.BatchNorm2d(planes * block.expansion),
            )
        layers = []
        layers.append(block(self.inplanes, planes, stride, downsample))
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.inplanes, planes))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x0 = self.relu(x)
        x = self.maxpool(x0)
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x4 = self.layer4(x3)
        return x4, x3, x2, x1, x0
```

程式碼 6. ResNetEncoder

實作了一個符合 ResNet34 架構的 影像特徵提取器。

透過多層殘差區塊逐層抽取不同階層的影像特徵，

最終輸出多組不同解析度的特徵圖，可供後續模組進行處理。

DecoderBlock

```
class DecoderBlock(nn.Module):
    """UNet 解碼器中的一個區塊"""
    def __init__(self, in_channels, out_channels):
        super(DecoderBlock, self).__init__()
        self.up = nn.ConvTranspose2d(in_channels, out_channels, kernel_size=2, stride=2)
        self.conv = nn.Sequential(
            conv3x3(out_channels * 2, out_channels),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            conv3x3(out_channels, out_channels),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )
    def forward(self, x, skip):
        x = self.up(x)
        # 處理因下採樣導致的尺寸不匹配問題
        # 如果 skip connection 的特徵圖比升採樣後的特徵圖大，則進行填充
        if skip.size(2) != x.size(2) or skip.size(3) != x.size(3):
            diffY = skip.size()[2] - x.size()[2]
            diffX = skip.size()[3] - x.size()[3]
            x = F.pad(x, [diffX // 2, diffX - diffX // 2,
                          diffY // 2, diffY - diffY // 2])

        x = torch.cat([x, skip], dim=1)
        x = self.conv(x)
        return x
```

程式碼 7. DecoderBlock

用於將編碼器中提取的深層特徵圖逐層進行空間解析度還原，
並透過 skip connection 融合對應層的淺層特徵圖。
這樣的設計有助於保留細節資訊，同時恢復空間結構。

ResNet34_UNet network

```
class ResNet34_UNet(nn.Module):
    def __init__(self, n_channels=3, n_classes=1):
        super(ResNet34_UNet, self).__init__()
        # 編碼器
        self.encoder = ResNetEncoder(BasicBlock, [3, 4, 6, 3], n_channels=n_channels)
        # 解碼器
        self.decoder4 = DecoderBlock(in_channels=512, out_channels=256)
        self.decoder3 = DecoderBlock(in_channels=256, out_channels=128)
        self.decoder2 = DecoderBlock(in_channels=128, out_channels=64)
        self.decoder1 = DecoderBlock(in_channels=64, out_channels=64)
        # 最終的升採樣和輸出層
        self.final_up = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2)
        self.final_conv = nn.Sequential(
            conv3x3(32, 32),
```

```
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            conv1x1(32, n_classes)
        )
    def forward(self, x):
        # 編碼器路徑
        e4, e3, e2, e1, e0 = self.encoder(x)
        # 解碼器路徑
        d4 = self.decoder4(e4, e3)
        d3 = self.decoder3(d4, e2)
        d2 = self.decoder2(d3, e1)
        d1 = self.decoder1(d2, e0)
        # 最終輸出
        out = self.final_up(d1)
        out = self.final_conv(out)
        return out
```

程式碼 8. ResNet34_UNet
結合 ResNet 編碼器與 U-Net 解碼器的模型架構。

原始 U-Net 以及部分 ResNet34_UNet 實作中，卷積層未使用 padding，導致每次卷積後特徵圖空間尺寸會縮小，跳接時必須手動裁切（crop）以對齊尺寸。所有 3×3 卷積層均設定 padding=1，確保輸入與輸出尺寸一致。此設計簡化了跳接的尺寸對齊問題，避免裁切，提高實作便利性。

由於輸入尺寸未必為 2 的冪次方倍數，經過多層下採樣與上採樣後，解碼器與編碼器跳接特徵圖仍可能存在尺寸差異。在解碼器中使用 PyTorch 的 F.pad 自動補齊上採樣特徵圖，避免因尺寸不符導致融合錯誤。

```
[Input] shape: torch.Size([1, 3, 572, 572])
--- Encoder Outputs ---
[e0] after initial conv: torch.Size([1, 64, 286, 286])
[e1] after layer1:       torch.Size([1, 64, 143, 143])
[e2] after layer2:       torch.Size([1, 128, 72, 72])
[e3] after layer3:       torch.Size([1, 256, 36, 36])
[e4] after layer4:       torch.Size([1, 512, 18, 18])
----------------------

--- Decoder Path ---
[d4] after decoder4(e4, e3): torch.Size([1, 256, 36, 36])
[d3] after decoder3(d4, e2): torch.Size([1, 128, 72, 72])
[d2] after decoder2(d3, e1): torch.Size([1, 64, 143, 143])
[d1] after decoder1(d2, e0): torch.Size([1, 64, 286, 286])
----------------------

--- Final Output ---
[out] after final_up:   torch.Size([1, 32, 572, 572])
[out] after final_conv: torch.Size([1, 1, 572, 572])
--------------------
```

圖 2. ResNet32_UNet 每一層特徵圖尺寸變化

1.2 Detail of Train / Evalulation / Inference

Train

Training.py

根據輸入參數選擇模型架構，包含：

- **U-Net**：經典的 encoder-decoder 架構。
- **ResNet34-UNet**：將 ResNet34 作為 encoder，結合 U-Net decoder，用以強化特徵提取能力。

設定訓練所需元件：

- **Optimizer**：AdamW，具備 L2 正規化（weight decay）與自適應學習率特性，有助於穩定且快速收斂。
- **Loss Function**：BCEWithLogitsLoss，數值穩定、適用於二元分類語意分割任務。
- **Scheduler**（學習率排程器，可選）

為了讓模型在訓練過程中能以更合適的步調學習，可以選擇不同的學習率調整策略。以下為支援的排程器說明：

排程器名稱	調整方式說明	適用情境與特點
cosine	學習率呈餘弦曲線下降，隨訓練輪數逐步變小	前期學得快，後期慢慢穩定，適合長時間訓練
onecycle	學習率先上升再下降，中間達到最高點	適合訓練 epoch 較少的情況，常用於微調（fine-tuning）
step	每隔固定輪數(如每 10 輪)將學習率乘上固定係數（如 0.1）	控制簡單，適合手動指定學習率下降時機
multistep	在特定的 epoch 點（如第 20 和 40 輪）手動設定學習率下降	適合已知學習率應該在何時調整的策略

訓練過程中輔以以下工具觀察訓練狀況：

- **tqdm**：即時顯示進度條與當前損失、Dice score、學習率。
- **wandb**（選用）：遠端記錄與可視化各項訓練指標。

訓練迴圈流程遵循四個步驟：

Forward → Calculate Loss → Backward → Update

每個 mini-batch 執行以下操作：

- **Forward**：輸入影像進行前向傳播，得到預測遮罩。
- **Calculate Loss**：使用 BCEWithLogitsLoss 計算模型預測與真實遮罩之間的差異。
- **Backward**：反向傳播誤差，計算參數梯度。
- **Update**：由 AdamW 更新模型權重。

```
for epoch in range(args.epochs):
    model.train() # 將模型設定為訓練模式
    total_loss = 0.0
    total_dice = 0.0

    # 使用 tqdm 顯示進度條
    pbar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{args.epochs}", leave=True)
    for batch in pbar:
        images = batch['image'].to(device)
        masks = batch['mask'].to(device)
        optimizer.zero_grad() # 清除舊的梯度
        outputs = model(images) # forward

        masks = masks.float()  # 確保 mask 的資料型別為 float, 以匹配模型的輸出
        loss = criterion(outputs, masks)
        loss.backward() # 反向傳播
        optimizer.step() # 更新權重

        # 如果使用 OneCycleLR, 則在每個 step 後更新學習率
        if args.scheduler == 'onecycle':
            scheduler.step()

        # 累加 loss 和 dice score
        total_loss += loss.item()
        total_dice += dice_score(outputs, masks)

        # 更新進度條的後綴訊息, 顯示即時的 loss 和平均 dice
        pbar.set_postfix(loss=f"{loss.item():.4f}", dice=f"{total_dice / (pbar.n + 1

    # 在 epoch 結束後, 更新 CosineAnnealingLR
    if args.scheduler == 'cosine':
        scheduler.step()

    # 計算整個 epoch 的平均 loss 和 dice
    avg_train_loss = total_loss / len(train_loader)
    avg_train_dice = total_dice / len(train_loader)

    print(f"Epoch {epoch+1} 完成 | 平均訓練損失: {avg_train_loss:.4f}, 平均訓練 Dice: {a
```

程式碼 9. Train 程式碼結構

wandb 視覺化顯示

```
# 初始化 wandb (如果可用且被啟用)
if args.use_wandb:
    if WANDB_AVAILABLE:
        try:
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            wandb.init(project=args.project, config=vars(args),
name=f'{args.model}_{timestamp}')
            print("Weights & Biases (wandb) 已成功啟用。")
        except Exception as e:
            print(f"初始化 wandb 時發生錯誤: {e}")
            print("將禁用 wandb 功能。請確認您已安裝 wandb (pip install wandb) 並已登
入 (wandb login)。")
            args.use_wandb = False
    else:
        print("警告: 您指定了 --use_wandb，但 wandb 套件未安裝。將禁用 wandb 功
能。")
        args.use_wandb = False
# 使用 wandb 紀錄
if args.use_wandb:
    wandb.log({
        'epoch': epoch + 1,
        'train_loss': avg_train_loss,
        'train_dice': avg_train_dice,
        'learning_rate': optimizer.param_groups[0]['lr']
    })
```

程式碼 10. Weights & Biases (wandb) 初始化，在每個 epoch 結束時，將訓練過程中數值儲存至 wandb，用於可圖形顯示。

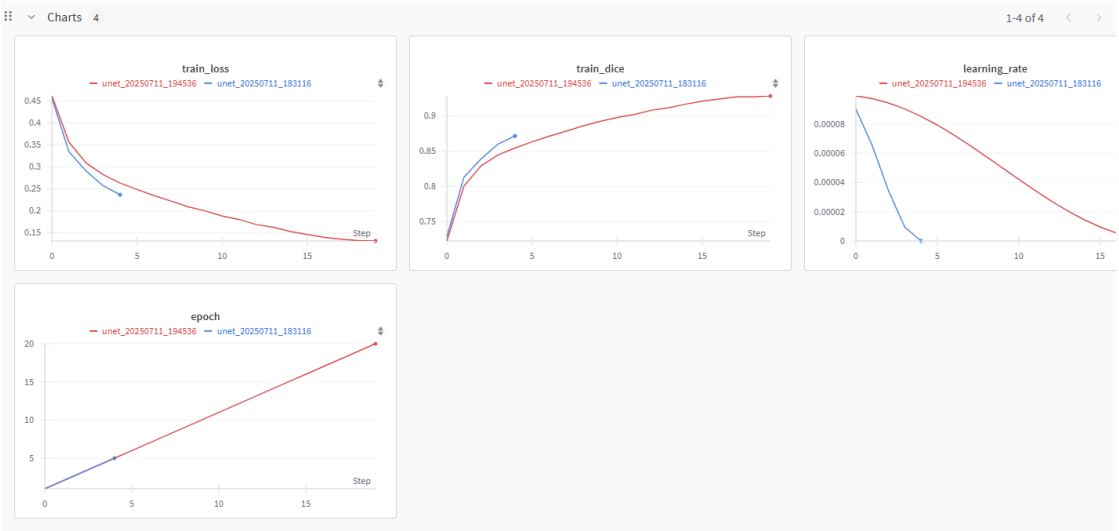


圖 3. Wandb 顯示訓練的數值

Loss function

BinaryCrossEntropy（BCE，二元交叉熵）

損失函數用於衡量二分類模型預測機率與實際標籤之間的差異，其定義如下

$$L = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})]$$

其中：

n: 樣本數

$y\_i \in \{0,1\}$: 第 i 筆真實標籤

$\hat{y_i} \in (0,1)$: 模型對第 i 筆預測為正類的機率(為 sigmoid 輸出)

在反向傳播時，需對每一筆預測值$\hat{y_i}$計算偏微分。推導如下：

$$\begin{aligned}\frac{\partial L}{\partial \hat{y}_i} &= -\frac{1}{n} \cdot \frac{\partial}{\partial \hat{y}_i}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] \\ &= -\frac{1}{n}\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right)\end{aligned}$$

因此，二元交叉熵的梯度公式為：

$$\boxed{\frac{\partial L}{\partial \hat{y}_i} = -\frac{1}{n}\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right)}$$

Evaluation

在訓練完成後使用驗證資料集對模型進行評估，評估流程不會進行梯度計算，僅執行前向傳播與性能計算，具體步驟如下：

模式切換與關閉梯度計算

模型切換至 eval 模式（model.eval()），並透過 torch.no_grad() 關閉梯度計算，

以降低記憶體使用並提升推論效率。

逐批次推論與 Dice 分數計算

使用 DataLoader 將驗證集分批讀入，並逐批進行下列操作：

- 載入輸入影像與對應的 Ground Truth 遮罩，搬移至指定運算裝置（CPU 或 GPU）。
- 輸入影像進行前向傳播，取得模型預測之 logits。
- 計算當前批次的 Dice 分數，並累加至紀錄列表。
- 利用 tqdm 顯示即時進度條與目前的平均 Dice 分數。

可視化預測結果（選用）

若提供輸出資料夾路徑，對前幾個批次部分影像儲存可視化結果，流程如下：

- 將預測 logits 經 sigmoid 轉為機率，再以閾值 0.5 二值化，產生預測遮罩。
- 每個批次擷取前數張影像，儲存其輸入影像、真實遮罩與預測遮罩的視覺化圖檔，方便後續人工檢查模型預測品質。

最終輸出

- 回傳或記錄整體平均 Dice 分數，作為模型效能評估指標。

```
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> python src/evaluate.py
使用裝置: cuda
資料集已存在。
驗證集大小: 368
從 saved_models/unet.pth 載入模型

評估完成！
驗證集上的平均 Dice 分數: 0.9165
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2>
```

圖 4. 驗證模型計算 Dice 分數

```
with torch.no_grad():
    # 使用 tqdm 顯示進度條
    pbar = tqdm(dataloader, desc="評估中", leave=False)
    for i, batch in enumerate(pbar):
        images = batch['image'].to(device)
        gt_masks = batch['mask'].to(device)

        # 前向傳播
        pred_logits = model(images)

        # 計算這個批次的平均 Dice 分數
        batch_dice = dice_score(pred_logits, gt_masks)
        all_dices.append(batch_dice)

        pbar.set_postfix(avg_dice=f"{(sum(all_dices) / len(all_dices)):.4f}")

        # 如果需要，儲存視覺化結果
        if output_dir:
            # 將預測的 logits 轉換為二元遮罩
            pred_masks = (torch.sigmoid(pred_logits) > 0.5).float()
            for j in range(images.shape[0]):
                # 只儲存前幾個批次的前幾張圖，避免產生過多檔案
                if i < 5 and j < 2:
                    save_figure(
                        images[j].cpu(),
                        gt_masks[j].cpu(),
                        pred_masks[j].cpu(),
                        os.path.join(output_dir, f"batch_{i}_img_{j}.png")
                    )

avg_dice = sum(all_dices) / len(all_dices) if all_dices else 0
return avg_dice, all_dices
```

程式碼 11. Evaluate 程式碼結構

Inference

本階段使用測試集評估模型效能，並視需要儲存視覺化結果。整體流程如下：

環境與資料準備：自動選擇裝置（GPU 優先），載入測試資料與模型權重。支援兩種模型架構：U-Net 與 ResNet34-UNet。

模型推論：關閉梯度計算，逐批進行前向傳播，並使用 dice_score 評估預測結果與真實遮罩的相似度。

視覺化與輸出（選用）：若指定輸出資料夾，儲存部分預測結果（最多前 5 個批次），並繪製 Dice 分數分布圖。

結果統計：回報整體平均 Dice 分數，作為模型在測試集上的最終表現指標。

```python
def inference(args):
    """
    在測試集上對已訓練的模型進行測試，回報平均 Dice 分數，並可選擇性地儲存視覺化結果。
    """
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f"使用裝置: {device}")

    if args.output_dir:
        import shutil
        if os.path.exists(args.output_dir):
            shutil.rmtree(args.output_dir)
        os.makedirs(args.output_dir)
        print(f"視覺化結果將儲存於: {args.output_dir}")

    if args.model_type == 'unet':
        model = UNet(n_channels=3, n_classes=1)
    elif args.model_type == 'resnet34_unet':
        model = ResNet34_UNet(n_channels=3, n_classes=1)

    model.load_state_dict(torch.load(args.model_path, map_location=device))
    model.to(device)
    model.eval()
    print(f"從 {args.model_path} 載入模型")

    test_dataset = load_dataset(args.data_path, mode='test', image_size=(args.image_size, args.image_size))
    test_loader = DataLoader(test_dataset, batch_size=args.batch_size, shuffle=False)
    print(f"測試集大小: {len(test_dataset)}")

    all_dices = []
    with torch.no_grad():
        for i, batch in enumerate(tqdm(test_loader, desc="測試中")):
            images = batch['image'].to(device)
            gt_masks = batch['mask'].to(device)

            pred_logits = model(images)

            dice = dice_score(pred_logits, gt_masks)
            all_dices.append(dice)

            if args.output_dir and i < 5:
                pred_masks = (torch.sigmoid(pred_logits) > 0.5).float()
                for j in range(images.shape[0]):
                    save_path = os.path.join(args.output_dir, f"batch_{i}_img_{j}.png")
                    save_visualization(images[j], gt_masks[j], pred_masks[j], save_path)

    avg_dice_score = sum(all_dices) / len(all_dices) if all_dices else 0

    print(f"\n測試完成！")
    print(f"測試集上的平均 Dice 分數: {avg_dice_score:.4f}")
```

程式碼 12. Inference 程式碼結構

1.3 Detail of utils

1. **D**ice score

我們採用 **Dice Score**（亦稱為 Dice Similarity Coefficient, DSC）作為主要的模型性能評估指標，數學定義如下：

$$DiceScore = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

- A：Ground Truth 遮罩中標記為前景（label = 1）的像素集合
- B：模型預測遮罩中預測為前景的像素集合
- |A∩B|：交集中的像素數量（即正確預測為前景的像素）

```python
with torch.no_grad():
    # --- Step 1: 移除單一通道維度，確保 shape 為 (B, H, W) ---
    if pred_mask.dim() == 4 and pred_mask.size(1) == 1:
        pred_mask = pred_mask.squeeze(1)
    if gt_mask.dim() == 4 and gt_mask.size(1) == 1:
        gt_mask = gt_mask.squeeze(1)

    # --- Step 2: 對預測結果進行 sigmoid 與閾值二值化 ---
    pred_bin = (torch.sigmoid(pred_mask) > threshold).float()
    gt_bin = (gt_mask > 0.5).float()

    # --- Step 3: 計算交集與聯集 (逐張圖片計算) ---
    # shape: (B,)
    intersection = (pred_bin * gt_bin).sum(dim=(1, 2))
    union = pred_bin.sum(dim=(1, 2)) + gt_bin.sum(dim=(1, 2))

    # --- Step 4: 計算 Dice Score 並取平均 ---
    dice = (2 * intersection + eps) / (union + eps)
    return dice.mean().item()
```

程式碼 13. Dice score 計算

2. Data Preprocessing

針對 Oxford-IIIT Pet Dataset 設計了高效且結構清晰的資料前處理流程，以便配合 Unet、ResNet34-UNet 架構的輸入與訓練需求。我們使用 **Albumentations 函式庫** 建立了一套高度可擴充的影像增強與正規化流程，具備以下優勢與改進：

2.1 尺寸調整與正規化

(有助於提升訓練穩定性與收斂速度)

```python
# 進行標準化（將像素值從 [0,255] 映射到標準分佈），使用 ImageNet 的平均值與標準差
A.Normalize(
    mean=[0.485, 0.456, 0.406],  # 三個通道的平均值（R, G, B）
    std=[0.229, 0.224, 0.225],  # 三個通道的標準差
    max_pixel_value=255.0  # 原始像素最大值，用於正規化
),
```

程式碼 14. 調整正規化

- 圖片與遮罩統一縮放至 256×256，確保模型下採樣後的空間維度為整數，避免 shape mismatch。
- 正規化（Normalization）：
  將影像像素值從 [0,255]縮放至 [0,1]，並使用 ImageNet 的均值與標準差進行標準化處理：

Mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]

2.2 訓練階段的資料增強

```
# 建立訓練階段的資料增強流程（transformation pipeline）
self.train_transform = A.Compose([

    # 將影像與遮罩調整為指定尺寸（例如 256x256）
    A.Resize(height=image_size[0], width=image_size[1]),

    # 以 50% 的機率進行水平翻轉，模擬左右對稱變化
    A.HorizontalFlip(p=0.5),

    # 以 30% 的機率進行隨機旋轉（範圍 ±15 度），增強對角度變化的魯棒性
    A.Rotate(limit=15, p=0.3),

    # 以 50% 的機率對影像進行顏色擾動（包含亮度、對比、飽和度變化）
    A.ColorJitter(p=0.5),
```

程式碼 15. 資料增強

使用 Albumentations 實作的 **資料增強策略** 包括：

- HorizontalFlip：隨機水平翻轉圖片與遮罩（50% 機率）。
- Rotate(limit=15)：隨機旋轉 ±15 度（30% 機率），模擬不同拍攝角度。
- ColorJitter：改變亮度、對比與飽和度（50% 機率），提升對光線變化的魯棒性(Robustness)。
- 所有轉換皆保證 **圖片與遮罩同步處理**，避免對齊錯位。

3. Analyze the experiment results

3.1 參數

模型架構 (Model Architecture)：

- 可選模型: UNet, ResNet34-UNet

資料處理 (Data Handling):

- 圖片尺寸 (Image Size): 256x256 pixels
- 批次大小 (Batch Size): 8
- 資料載入線程數 (Num Workers): 2

優化器與損失函數 (Optimizer & Loss Function):

- 優化器 (Optimizer): AdamW
- 損失函數 (Loss Function): BCEWithLogitsLoss

訓練週期與學習率 (Epochs & Learning Rate):

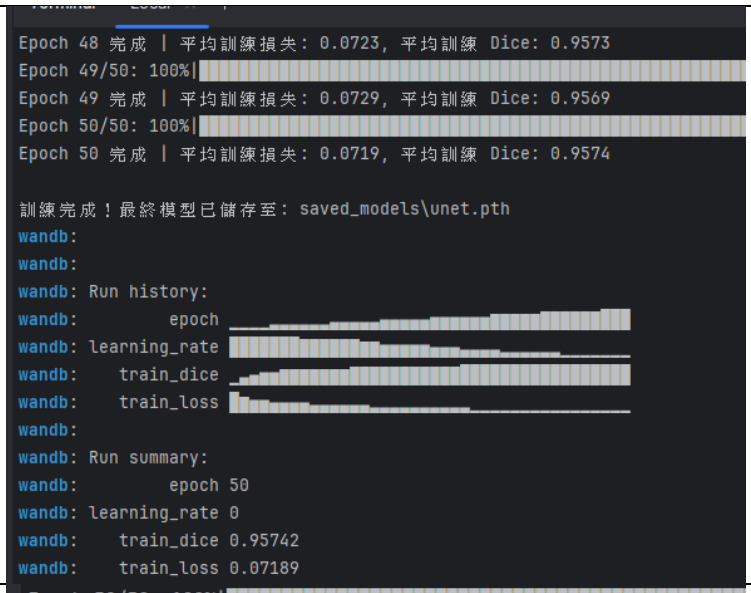
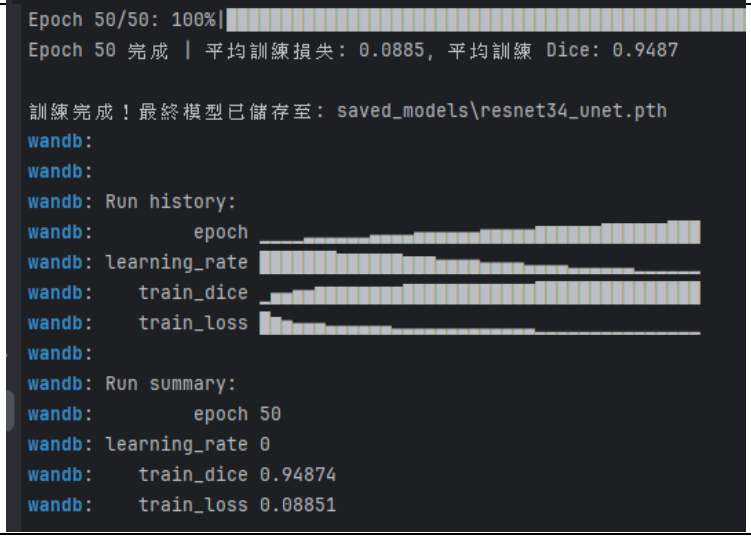
總訓練週期 (Epochs): 20
初始學習率 (Learning Rate): 1e-4 (0.0001)
權重衰減 (Weight Decay): 1e-4 (0.0001)

學習率排程器 (Learning Rate Scheduler):

可選策略: none, cosine, onecycle, step, multistep

3.2 實驗結果

3.2.1 Train

Unet (train)	
ResNet34_Unet (train)	

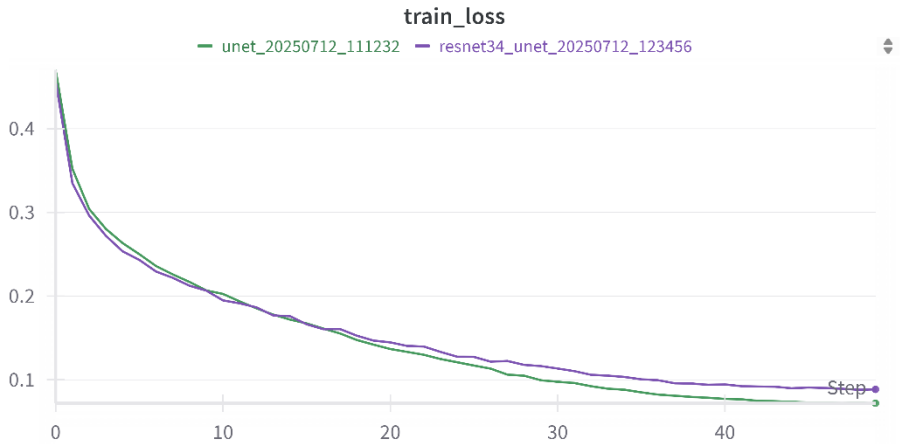


圖 5. Train loss

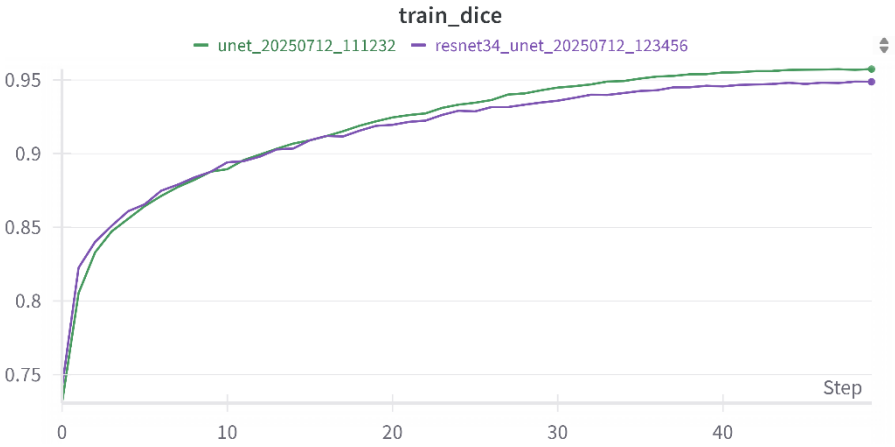


圖 6. Train dice

在使用 Oxford-IIIT Pet 資料集進行語意分割訓練，比較兩種模型架構 UNet 與 ResNet34-UNet 訓練成效，依據 **Train Dice** 與 **Train Loss** 進行分析。

訓練最終結果：

模型	平均訓練 Dice	平均訓練 Loss
UNet	0.9574	0.0719
ResNet34-UNet	0.9487	0.0885

分析說明：

- **Dice 分數方面**，UNet 模型的最終 Dice 為 0.9574，略高於 ResNet34-UNet 的 0.9487，代表其對前景與背景的區別能力稍佳。
- **Loss 值方面**，UNet 的最終訓練 Loss 為 0.0719，亦優於 ResNet34-UNet 的 0.0885，顯示其預測更為穩定、誤差較小。
- **整體而言**，UNet 在訓練集上的學習效率與收斂表現皆優於 ResNet34-UNet，可能與其架構較為簡潔、避免過度擬合有關。

3.2.2 Evaluate

unet	(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> python src/evaluate.py 使用裝置： cuda 資料集已存在。 驗證集大小： 368 從 saved_models/unet.pth 載入模型 評估完成！ 驗證集上的平均 Dice 分數： 0.9254 Dice 分數分佈圖已儲存至： eval_results/unet_viz\dice_distribution.png (dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> SS
ResNet34_Unet	(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> python src/evaluate.py 使用裝置： cuda 資料集已存在。 驗證集大小： 368 從 saved_models/resnet34_unet.pth 載入模型 評估完成！ 驗證集上的平均 Dice 分數： 0.9159 Dice 分數分佈圖已儲存至： eval_results/resnet_viz\dice_distribution.png (dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2>

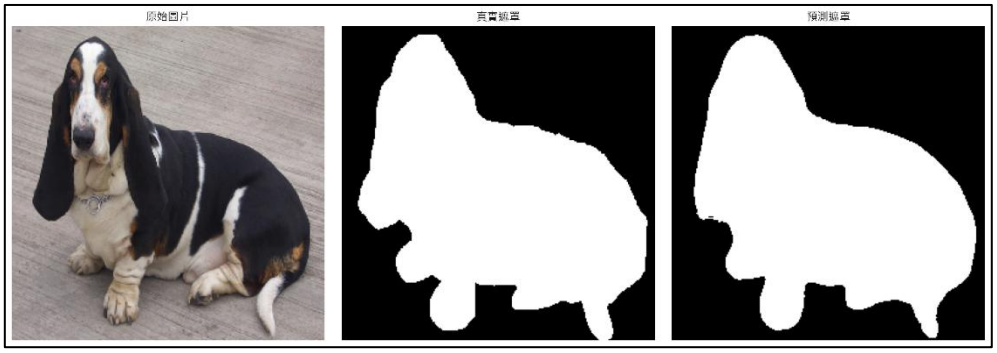


圖 7. Evaluate (unet)驗證的圖片

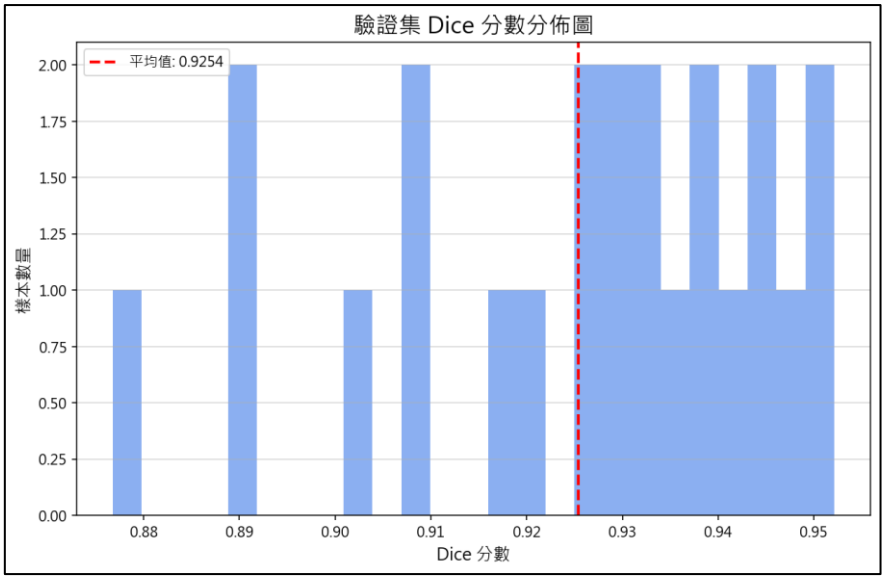


圖 8. Evaluate (unet)分數分布圖

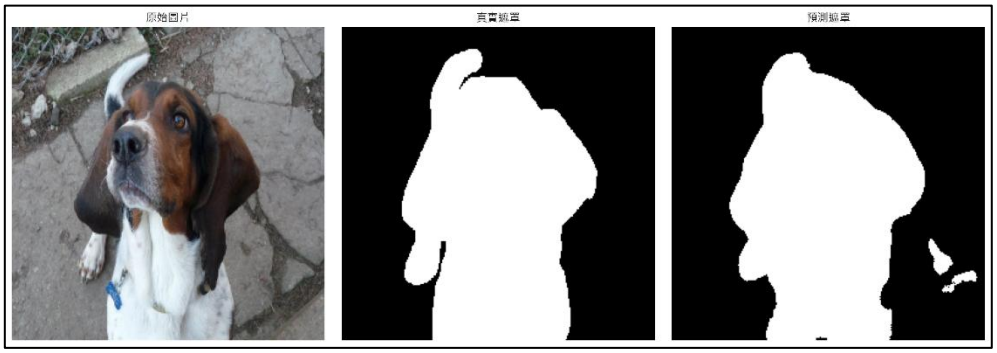


圖 9. Evaluate (resnet34_unet)驗證的圖片

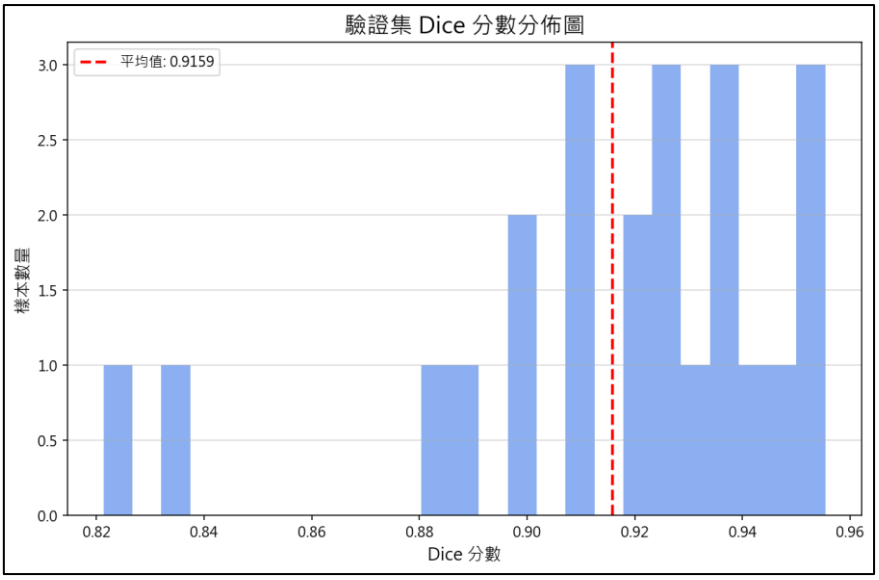


圖 10. Evaluate (resnet34_unet)分數分布圖

3.2.3 Inference

unet	(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> python src/inf 使用裝置： cuda 視覺化結果將儲存於： inference_results/unet_viz 從 saved_models/unet.pth 載入模型 資料集已存在。 測試集大小： 3669 測試中： 100%| 測試完成！ 測試集上的平均 Dice 分數： 0.9256 Dice 分數分佈圖已儲存至： inference_results/unet_viz\dice_distribution.png
ResNet34_Unet	(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2> python src/inference.py viz 使用裝置： cuda 視覺化結果將儲存於： inference_results/resnet_viz 從 saved_models/resnet34_unet.pth 載入模型 資料集已存在。 測試集大小： 3669 測試中： 100%| 測試完成！ 測試集上的平均 Dice 分數： 0.9258 Dice 分數分佈圖已儲存至： inference_results/resnet_viz\dice_distribution.png

圖 11. Inference (unet)測試圖片

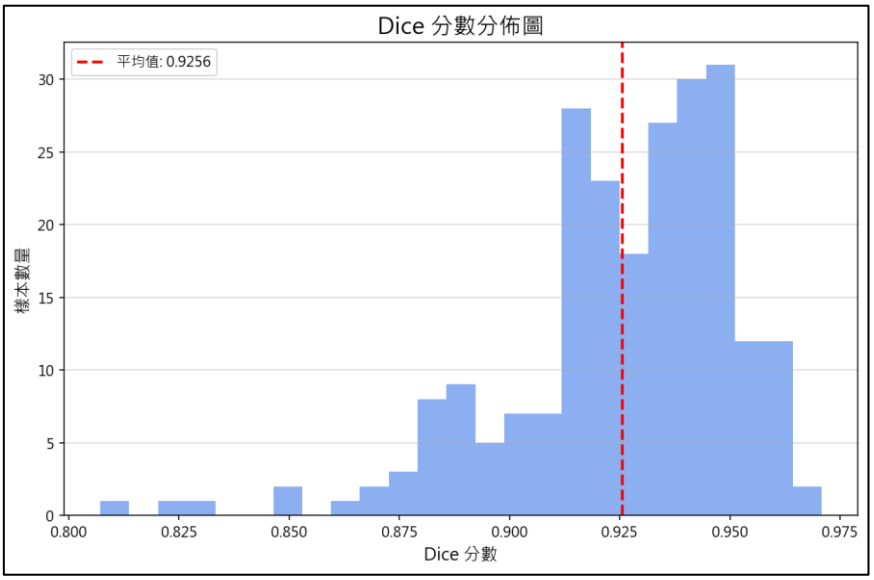


圖 12. Inference (unet)分數分布圖

圖 13. Inference (resnet34_unet)測試圖片

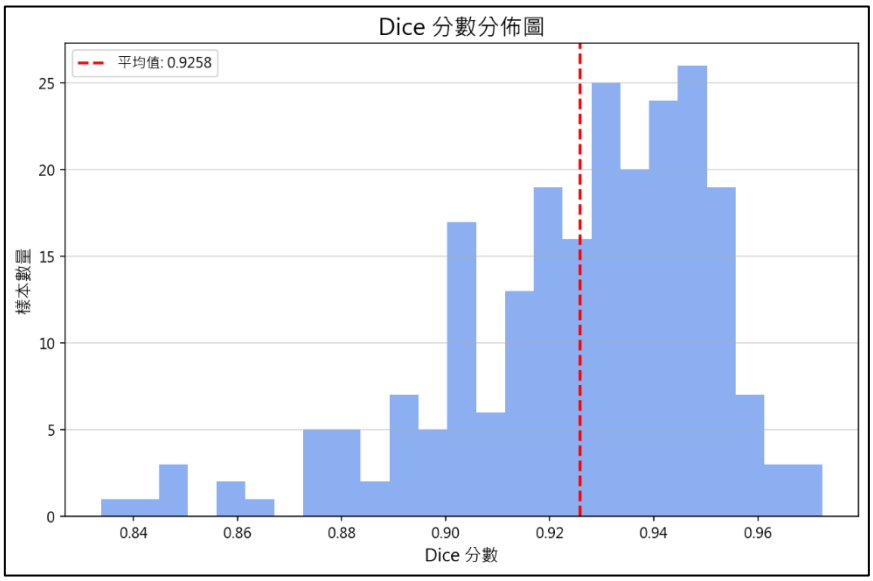


圖 14. Inference (resnet34_unet)分數分布圖 S

4. Execution steps

4.1 requirement.txt

透過 anaconda prompt 輸入”pip freeze > requirements.txt”導出環境

```
(base) C:\Users\krameri120>activate dlp

(dlp) C:\Users\krameri120>cd C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2

(dlp) C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2>pip freeze > requirements.txt

(dlp) C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab2>
```

4.2指令範例

4.2.1 訓練指令 unet

訓練 (20 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model unet --epochs 20
訓練 (50 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model unet --epochs 50
訓練-wandb (20 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model unet --epochs 20 --use_wandb
訓練-wandb (50 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model unet --epochs 50 --use_wandb

4.2.2 訓練指令 resnet34_unet

訓練(20 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model resnet34_unet --epochs 20
訓練(50 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model resnet34_unet --epochs 50
訓練-wandb (20 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model resnet34_unet --epochs 20 --use_wandb
訓練-wandb (50 epochs)	python src/train.py --data_path dataset/oxford-iiit-pet --model resnet34_unet --epochs 50 --use_wandb

4.2.3 驗證指令 unet

僅計算 Dice 分數	python src/evaluate.py --model_path saved_models/unet.pth --data_path dataset/oxford-iiit-pet --model_type unet
計算分數 儲存視覺化結果	python src/evaluate.py --model_path saved_models/unet.pth --data_path dataset/oxford-iiit-pet --model_type unet --output_dir eval_results/unet_viz

4.2.4 驗證指令 resnet34_unet

僅計算 Dice 分數	python src/evaluate.py --model_path saved_models/resnet34_unet.pth --data_path dataset/oxford-iiit-pet --model_type resnet34_unet
計算分數 儲存視覺化結果	python src/evaluate.py --model_path saved_models/resnet34_unet.pth --data_path dataset/oxford-iiit-

	pet --model_type resnet34_unet --output_dir eval_results/resnet_viz

4.2.5 測試指令 unet

僅測試	python src/inference.py --model_path saved_models/unet.pth --data_path dataset/oxford-iiit-pet --model_type unet
測試 儲存視覺化結果	python src/inference.py --model_path saved_models/unet.pth --data_path dataset/oxford-iiit-pet --model_type unet --output_dir inference_results/unet_viz

4.2.6 測試指令 resnet34_unet

僅測試	python src/inference.py --model_path saved_models/resnet34_unet.pth --data_path dataset/oxford-iiit-pet --model_type resnet34_unet
測試 儲存視覺化結果	python src/inference.py --model_path saved_models/resnet34_unet.pth --data_path dataset/oxford-iiit-pet --model_type resnet34_unet --output_dir inference_results/resnet_viz

5. Discussion

針對寵物影像語意分割作業，說明餘弦學習率調整策略的原理，並說明本專案實作中具有代表性的設計與技術。

5.1 餘弦學習率調整策略

Cosine Annealing 學習率調整策（torch.optim.lr_scheduler.CosineAnnealingLR）是一種先進的學習率調控技術。與傳統的階梯式（Step Decay）不同，它會根據餘弦函數的變化趨勢，平滑地調整學習率。

每一個 epoch 的學習率lr_t依據以下公式計算：

$$lr_t = lr_{min} + 0.5 \times (lr_{max} - lr_{min}) \times \left(1 + cos(T_{cur}/T_{max} \times \pi)\right)$$

lr_{max}：初始學習率（由 Optimizer 設定）

lr_{min}：最小學習率（常設為 0）

T_{max}：完整一個週期的 epoch 數

T_cur：當前 epoch

這樣的設計使學習率從較高的值平滑下降至最小值，若搭配「重啟機制」（cosine annealing with restarts），則可重新開始下一個循環，進一步提升訓練穩定性與效果。

- **提升收斂品質**：初期高學習率有助於快速探索，後期低學習率可精細調整模型參數。
- **跳出局部最小值**：若加入重啟策略，有助模型跳脫訓練早期陷入的次佳解。
- **穩健性高**：對於初始學習率的敏感度較低，參數選擇更為寬容。

5.2 資料切分策略

為了確保模型訓練的可靠性與評估的公正性，本專案採用了嚴謹的資料切分策略，並於 oxford_pet.py 中的 OxfordPetDataset 類別實作。

- **測試集**：使用官方提供的 test.txt 檔案作為測試資料，確保最終評估指標能與其他研究成果進行公平比較。
- **訓練與驗證集**：從官方提供的 trainval.txt 檔案中，依照 **9:1** 的比例進行劃分。實作方式為 i % 10 != 0，即將 90% 的資料分配給訓練集，其餘 10% 分配給驗證集。

5.3 遮罩（Mask）前處理

Oxford-IIIT Pet Dataset 中的原始遮罩（mask）為三分類的 Trimap：前景（1）、背景（2）、邊界（3）。為了將任務轉為二元語意分割，我們在 _preprocess_mask 函式中進行以下處理：

- **前景與邊界合併**：將像素值為 1（前景）與 3（邊界）的區域皆標記為 1，視為目標物件。
- **背景處理**：將像素值為 2 的區域標記為 0，視為背景。

最終產生的遮罩為僅含 0 與 1 的二元分類圖，使模型能明確學習物件區域與背景的區分。

5.4 資料增強（Data Augmentation）

為了提升模型的泛化能力，我們利用 Albumentations 對訓練增強。

- Resize：將影像尺寸統一為模型所需大小。
- HorizontalFlip：隨機水平翻轉影像，加強模型對左右對稱特徵的辨識能力。
- Rotate：隨機旋轉角度，提升模型對角度變化的適應性。
- ColorJitter：隨機改變亮度、對比、飽和度，模擬不同光源條件。
- Normalize：以 ImageNet 的平均值與標準差對影像標準化，加速模型訓練與收斂。
- ToTensorV2：將 NumPy 陣列轉為 PyTorch 張量，便於輸入模型訓練流程。

6. Demo Video Link

Youtube link：https://youtu.be/wc-73M0jZAo

7. 參考資料

1. https://github.com/jayin92/NYCU-deep-learning/tree/main/lab02
   (模型架構與訓練流程設計)
2. https://github.com/hank891008/Deep-Learning/tree/main/Lab3
   (載入與模型訓練的程式架構)
3. https://github.com/lolainta/NYCU-DL-2025-Spring/tree/1ac00a6153f8b44484e5972270490499053c39eb/hw2
   (資料處理與模型評估方法)
4. https://github.com/milesial/Pytorch-UNet
   (unet 參考資料)
5. https://blog.csdn.net/weixin_48524215/article/details/139814345
   (unet 參考資料)

6. https://blog.csdn.net/weixin_45144684/article/details/125168347
(resnet 參考資料)
7. https://github.com/albumentations-team/albumentations_examples/blob/main/notebooks/pytorch_semantic_segmentation.ipynb
(Albumentations 工具資料增強參考資料)
8. https://blog.csdn.net/qq_40507857/article/details/112791111
(wandb 視覺化工具)
9. https://blog.csdn.net/weixin_43913261/article/details/124687789
(anaconda 導出 requirement.txt)
10. https://blog.csdn.net/qq_43426908/article/details/125019923
(學習率調整)

使用的 AI 工具：

- ✧ **ChatGPT-4o**（由 OpenAI 提供）
  用於問答、報告草稿撰寫、英文轉中文及論文說明。
- ✧ **Google AI Studio**
  作為補充問答工具，用於模型概念、架構或函式撰寫思路輔助。
- ✧ **VSCode + Google Gemini 2.5 Pro**
  協助撰寫與補全 Python 程式碼，並支援簡易 debug。
- ✧ **VSCode GitHub Copilot（GPT-4o）**
  用於實作過程中進行即時程式補全、語法建議與除錯輔助。