Machine Learning HW2

N26130891 林以諾

(一)Model Architecture

本次實作的語意分割模型基於改良版的 UNet 架構，並結合 SE (Squeeze-and-Excitation) Block，提升模型的全局特徵學習能力及分割效果。下述內容詳細說明模型的架構設計。

1. Encoder Path (Downsampling)

- 功能：Encoder Path 負責提取輸入圖像的多層次特徵，逐步減少空間分辨率並增強通道特徵。
- 組成模組：
    - DoubleConv：

      由兩次卷積操作組成，每次卷積後接 Batch Normalization 和 ReLU 激活函數。提取局部特徵，同時保留重要細節。

    - MaxPooling：

      進行空間壓縮操作，減少特徵圖尺寸（如 256×256 到 128×128），提高計算效率。

- 輸出示例：
    - 初始輸入 256×256→卷積後通道數增加至 64，空間維度保持不變。
    - 通過 MaxPooling 後，空間尺寸降為 128×128

2. Bottleneck with SE Block

- 功能：Bottleneck 是模型的核心，用於提取輸入圖像的全局特徵。
- SE Block 設計：
    - 全局平均池化：通過空間壓縮將特徵圖轉換為通道級全局信息。
    - 全連接層：重新分配通道權重，增強關鍵特徵的學習。
- 作用：

加強 Bottleneck 部分的通道選擇能力，讓模型更關注對語意分割重要的特徵。

3. Decoder Path (Upsampling)

- 功能：逐步恢復圖像的空間分辨率，並通過跳躍連接補充細節。
- 組成模組：
    - Up：
        - 上採樣模組透過雙線性插值或反卷積增加空間分辨率。
        - 並與對應的 Encoder Path 層輸出進行拼接（Skip Connection），融合低層細節和高層語意信息。
    - DoubleConv：
        - 解碼後的特徵進一步卷積，恢復細節特徵。
- 輸出示例：

空間尺寸由 128×128 恢復成 256×256。

4. Output Layer

- 輸出語意分割結果。
- 組成模組：

單層卷積（OutConv）：將特徵圖映射到 13 個類別（對應數據集中的語意分割標籤）。

- 輸出示例：

經過解碼後，特徵圖尺寸保持 256×256，輸出通道數為 13。

5. 模型架構總覽

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 256]           1,792
       BatchNorm2d-2         [-1, 64, 256, 256]             128
              ReLU-3         [-1, 64, 256, 256]               0
            Conv2d-4         [-1, 64, 256, 256]          36,928
       BatchNorm2d-5         [-1, 64, 256, 256]             128
              ReLU-6         [-1, 64, 256, 256]               0
        DoubleConv-7         [-1, 64, 256, 256]               0
         MaxPool2d-8         [-1, 64, 128, 128]               0
            Conv2d-9        [-1, 128, 128, 128]          73,856
      BatchNorm2d-10        [-1, 128, 128, 128]             256
             ReLU-11        [-1, 128, 128, 128]               0
           Conv2d-12        [-1, 128, 128, 128]         147,584
      BatchNorm2d-13        [-1, 128, 128, 128]             256
             ReLU-14        [-1, 128, 128, 128]               0
       DoubleConv-15        [-1, 128, 128, 128]               0
             Down-16        [-1, 128, 128, 128]               0
        MaxPool2d-17          [-1, 128, 64, 64]               0
           Conv2d-18          [-1, 256, 64, 64]         295,168
      BatchNorm2d-19          [-1, 256, 64, 64]             512
             ReLU-20          [-1, 256, 64, 64]               0
           Conv2d-21          [-1, 256, 64, 64]         590,080
      BatchNorm2d-22          [-1, 256, 64, 64]             512
...
Forward/backward pass size (MB): 1094.02
Params size (MB): 120.25
Estimated Total Size (MB): 1215.03
```

(二)Problems and Solutions

1. 類別不均衡問題

- 在語意分割任務中，數據集中不同類別的像素數量分佈不均，在此次實作中發現 CLASS4,5,6,12 就相對較少。這可能導致模型更偏向學習主要類別，而忽略稀疏類別和小目標。
- 解決方案：

使用混合損失函數結合兩種不同的損失計算方式：

交叉熵損失 (Cross-Entropy Loss) 強調全局像素分類準確率。

Dice 損失 (Dice Loss) 強化模型對小目標及稀疏類別的分割性能。

```
def mixed_loss(outputs, targets, ce_weight=0.7, dice_weight=0.3):
    cross_entropy = nn.CrossEntropyLoss()(outputs, targets)
    dice_loss = DiceLoss()(outputs, targets)
    return ce_weight * cross_entropy + dice_weight * dice_loss
```

2. 特徵提取能力不足

- 統的 UNet 模型在 Bottleneck 部分缺乏全局特徵學習能力，可能導致模型在複雜場景下無法有效區分小目標與背景。
- 解決方案：
    - 在模型的 Bottleneck 階段加入 SE Block (Squeeze-and-Excitation Block)，強化通道特徵的學習能力：
        - 通道權重調整：學習每個通道的重要性，對高權重通道進行增強，抑制低權重通道。
        - 全局上下文捕捉：通過全局平均池化提取全局信息。

```
class SEBlock(nn.Module):
    def __init__(self, in_channels, reduction=16):
        super(SEBlock, self).__init__()
        self.global_pool = nn.AdaptiveAvgPool2d(1)
        self.fc1 = nn.Conv2d(in_channels, in_channels // reduction, kernel_size=1)
        self.relu = nn.ReLU(inplace=True)
        self.fc2 = nn.Conv2d(in_channels // reduction, in_channels, kernel_size=1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        se = self.global_pool(x)
        se = self.relu(self.fc1(se))
        se = self.sigmoid(self.fc2(se))
        return x * se
```

3. 過擬合現象

- 在模型訓練中，過擬合會導致模型過度學習訓練數據，從而降低在驗證集和測試集上的性能。
- 解決方案：
    - 採用 數據增強技術 增強模型的泛化能力：
        - 隨機水平翻轉 (RandomHorizontalFlip) 增加數據多樣性。
        - 標準化 (Normalize) 減少圖像亮度和顏色差異對模型學習的影響。
    - Dropout 是一種正則化技術，用於隨機丟棄部分神經元的輸出，防止模型過度依賴某些特徵。在模型中添加 Dropout，可以有效減少過擬合。
        - 在 Bottleneck 部分 應用 Dropou

```
self.dropout = nn.Dropout(p=0.4)  # 丟棄率為 40%
x_bottleneck = self.dropout(x_bottleneck)
```

        - 在 DoubleConv 模塊 中添加 Dropout

```
self.double_conv = nn.Sequential(
    nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.3),  # 丟棄率為 30%
    nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.3)
)
```

    - 使用 驗證集 監控模型的泛化性能，並分割訓練數據集為 90% 訓練和 10% 驗證。

```
train_transforms = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])
```

(三)Methods to improve Accuracy

1. 增強模型架構

改進編碼器與解碼器結構：

在編碼器和解碼器中使用 DoubleConv 模塊，結合 Batch Normalization 和 ReLU 激活函數，提升特徵提取能力並穩定梯度流動。

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.double_conv = 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)
        )
    def forward(self, x):
        return self.double_conv(x)
```

2. 使用混合精度訓練

- 混合精度訓練減少了顯存占用，允許使用更大的批次大小，減少訓練資源需求，同時提升模型的訓練穩定性進一步提升訓練效率。

```
scaler = torch.cuda.amp.GradScaler()  # 混合精度工具
with torch.cuda.amp.autocast():
    logits = model(images)
    loss = mixed_loss(logits, masks)
scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

3. 學習率調度策略

- 採用餘弦退火學習率調度器，隨訓練進程逐步降低學習率，提升模型穩定性，提高了模型收斂速度，同時避免過擬合。

```
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10, eta_min=1e-6)
```

```
scheduler.step()
```

(四)Analysis of Results

a. 訓練與驗證表現

1. 整體訓練表現
   模型在 50 個 epoch 中表現穩定，訓練損失從 0.4553 降至 0.3306，訓練準確率從 89.55% 提升至 93.64%。
   驗證損失在前幾個 epoch 快速下降，並在 第 50 個 epoch 達到最低值 0.2818，驗證準確率也達到 92.36%。

2. 正則化與數據增強的效果

   - 使用 隨機水平翻轉 和 標準化 增強了數據多樣性，有效提高模型的泛化能力。
   - Dropout 的應用成功抑制了過擬合，保證了驗證性能穩定。

3. 模型收斂性

   - 驗證損失與準確率曲線顯示模型逐步收斂，並在最後幾個 epoch 實現了穩定的高性能。
   - 最佳模型權重儲存在 best_model.pth。

b. 關鍵觀察

1. 穩定提升
   模型在訓練與驗證過程中，表現出良好的穩定性，訓練與驗證損失曲線均平滑下降。

2. 局部波動
   雖然大部分 epoch 表現良好，但部分驗證損失（如第 7、13 和 22 個 epoch）有小幅波動，可能與驗證數據隨機性或模型對部分樣本的過擬合有關。

3. 最佳表現
   在第 50 個 epoch，模型達到最優驗證損失 0.2818，驗證準確率 92.36%，顯示了穩健的泛化能力。

c. 關鍵 epoch 紀錄

epoch	Train loss	Train acc	Val loss	Val acc
1	0.4553	89.55%	0.6048	82.98%
25	0.3812	91.94%	0.4664	87.86%
30	0.3511	93.05%	0.3670	90.52%
50	0.3306	03.64%	0.2818	92.36%

d. 結論

- 模型透過數據增強和正則化技術，顯著減少了過擬合問題，實現穩定的性能提升。
- 混合損失函數（交叉熵 + Dice + IoU）幫助模型在類別不均衡情況下，取得更好的分割效果。
- 最終的驗證準確率達92.36%，證明模型對驗證集和可能的測試集都有良好的適應能力。