Lab3: Binary Semantic Segmentation

313551104 黃暐洺

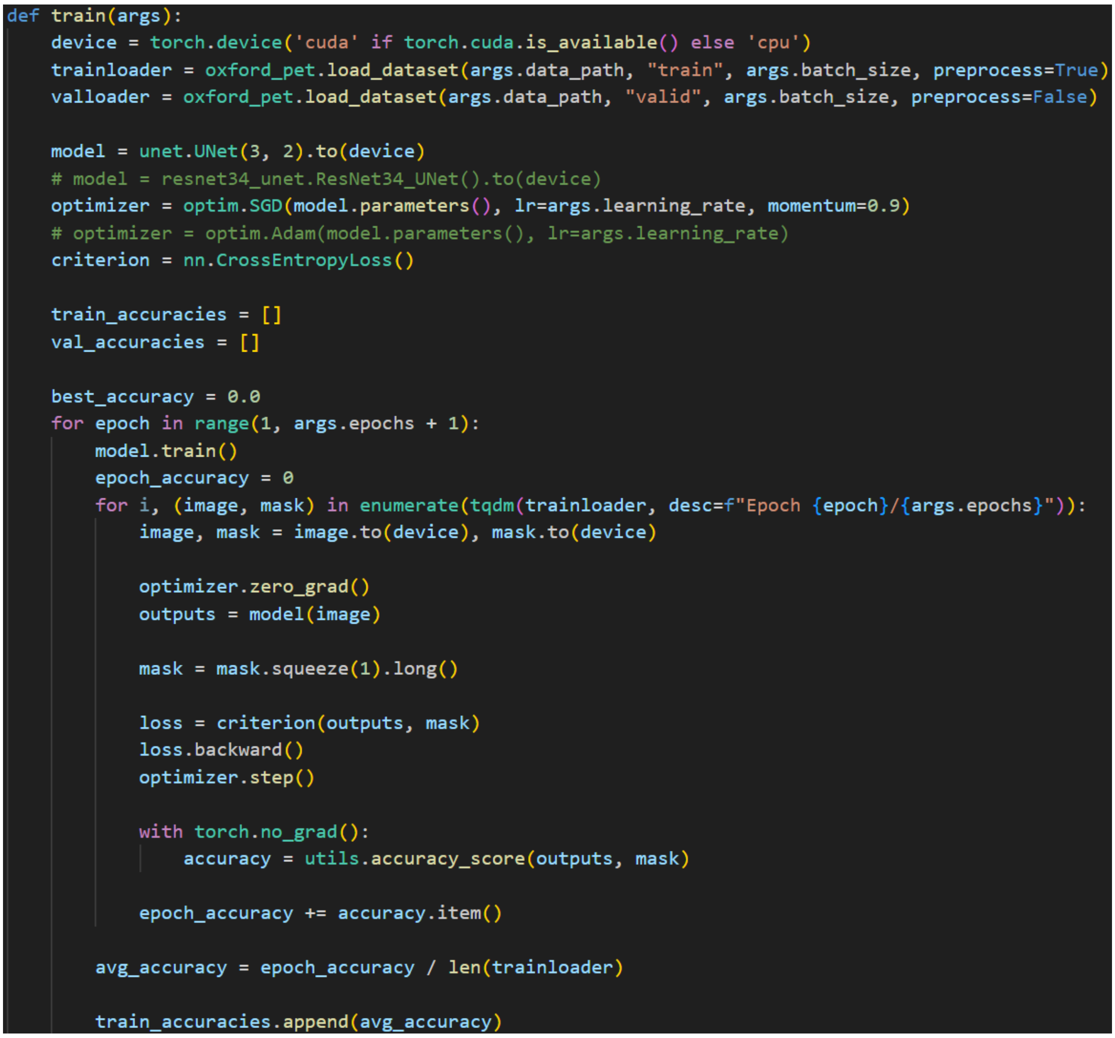
1. Overview of your lab 3 (10%)

   本次實驗實現語意分割的任務，使用 Oxford-IIIT Pet Dataset 訓練和評估，並實作了兩種模型架構：UNet 和 ResNet34_UNet，之後比較這兩種模型的性能，並分析實驗結果。

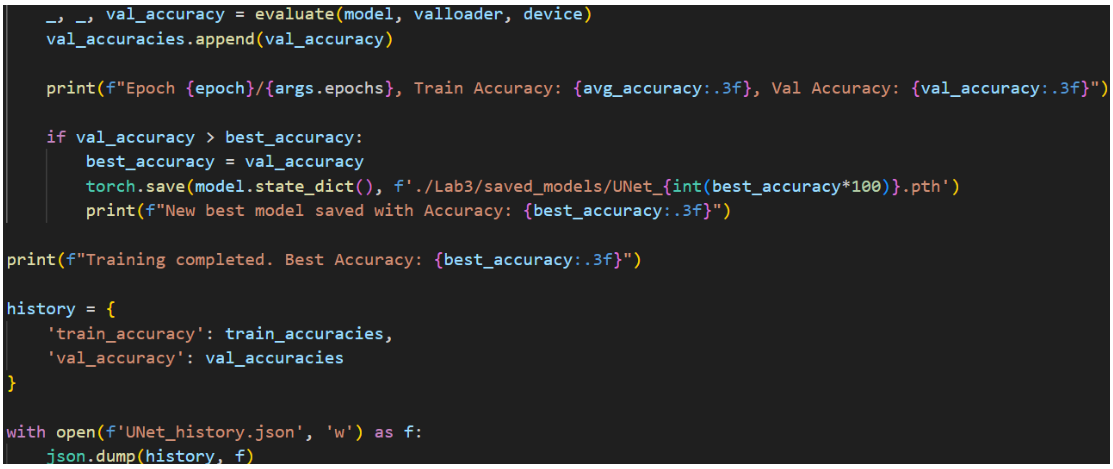
2.Implementation Details (30%)

➢ Details of your training, evaluating, inferencing code

train.py



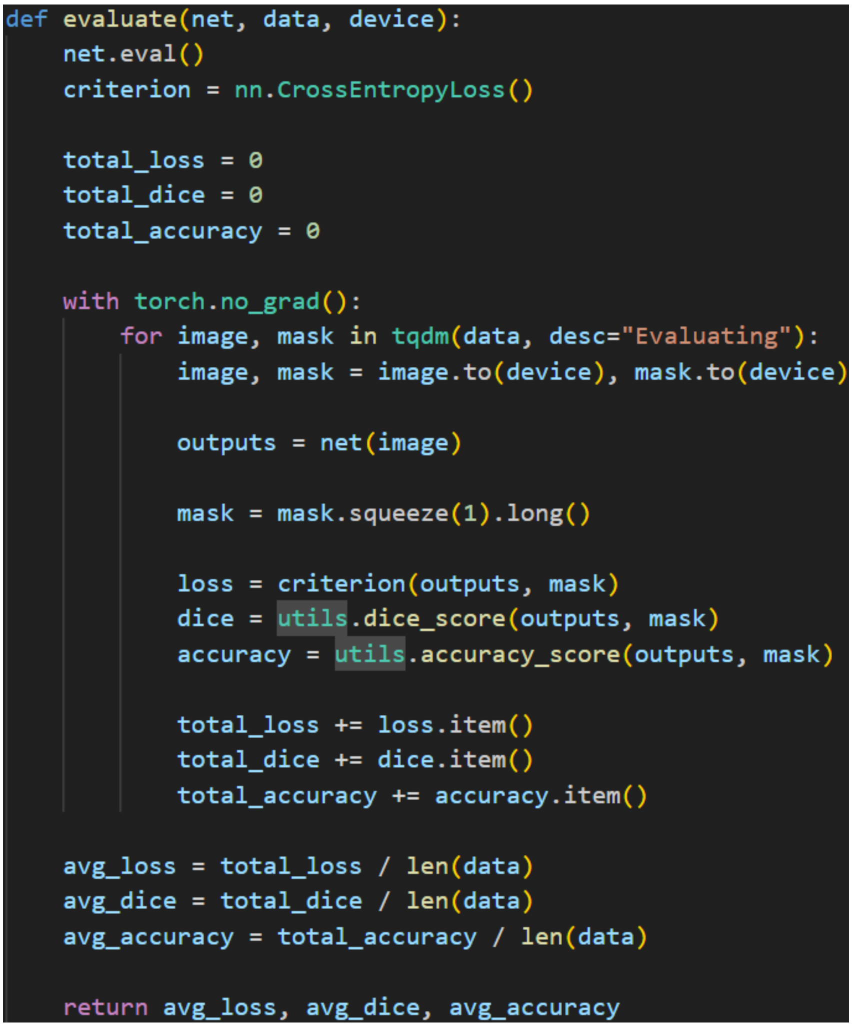
```
def train(args):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    trainloader = oxford_pet.load_dataset(args.data_path, "train", args.batch_size, preprocess=True)
    valloader = oxford_pet.load_dataset(args.data_path, "valid", args.batch_size, preprocess=False)

    model = unet.UNet(3, 2).to(device)
    # model = resnet34_unet.ResNet34_UNet().to(device)
    optimizer = optim.SGD(model.parameters(), lr=args.learning_rate, momentum=0.9)
    # optimizer = optim.Adam(model.parameters(), lr=args.learning_rate)
    criterion = nn.CrossEntropyLoss()

    train_accuracies = []
    val_accuracies = []

    best_accuracy = 0.0
    for epoch in range(1, args.epochs + 1):
        model.train()
        epoch_accuracy = 0
        for i, (image, mask) in enumerate(tqdm(trainloader, desc=f"Epoch {epoch}/{args.epochs}")):
            image, mask = image.to(device), mask.to(device)

            optimizer.zero_grad()
            outputs = model(image)

            mask = mask.squeeze(1).long()

            loss = criterion(outputs, mask)
            loss.backward()
            optimizer.step()

            with torch.no_grad():
                accuracy = utils.accuracy_score(outputs, mask)

            epoch_accuracy += accuracy.item()

        avg_accuracy = epoch_accuracy / len(trainloader)

        train_accuracies.append(avg_accuracy)
```



```
    _, _, val_accuracy = evaluate(model, valloader, device)
    val_accuracies.append(val_accuracy)

    print(f"Epoch {epoch}/{args.epochs}, Train Accuracy: {avg_accuracy:.3f}, Val Accuracy: {val_accuracy:.3f}")

    if val_accuracy > best_accuracy:
        best_accuracy = val_accuracy
        torch.save(model.state_dict(), f'./Lab3/saved_models/UNet_{int(best_accuracy*100)}.pth')
        print(f"New best model saved with Accuracy: {best_accuracy:.3f}")

print(f"Training completed. Best Accuracy: {best_accuracy:.3f}")

history = {
    'train_accuracy': train_accuracies,
    'val_accuracy': val_accuracies
}

with open(f'UNet_history.json', 'w') as f:
    json.dump(history, f)
```

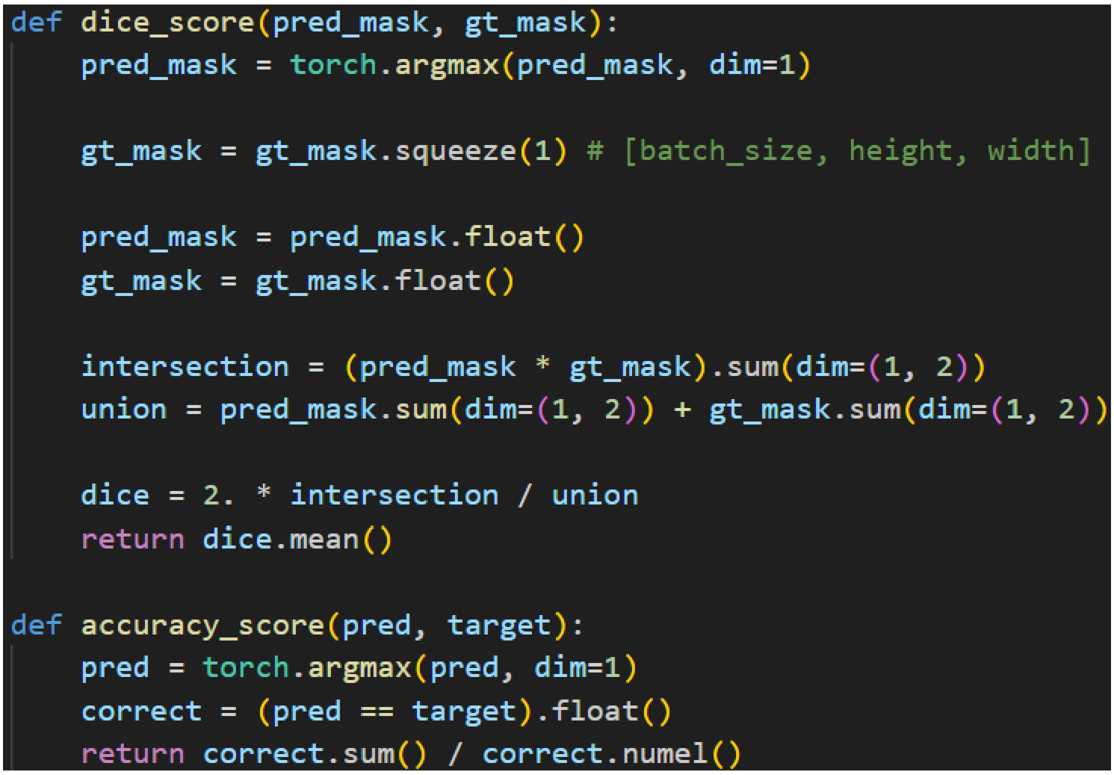
跟之前的 Lab 沒有甚麼差別，優化器有用 Adam 跟 SGD，做完 Epoch 會跑一次驗證集看準確率，保存準確率最佳的模型。

evaluate.py



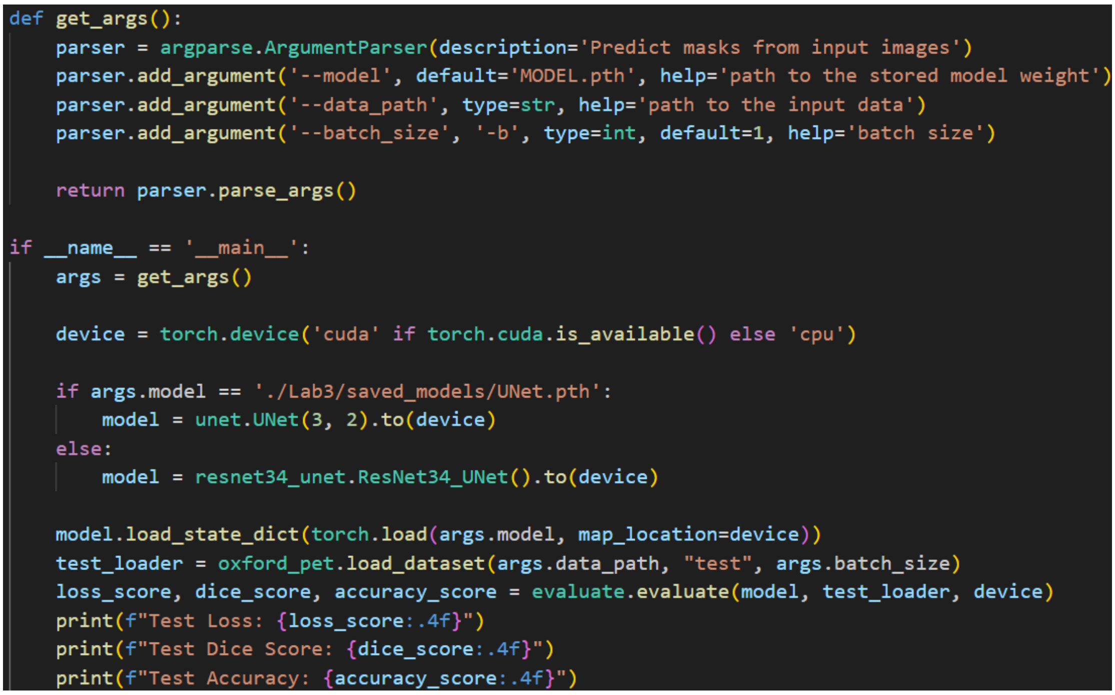
```
def evaluate(net, data, device):
    net.eval()
    criterion = nn.CrossEntropyLoss()

    total_loss = 0
    total_dice = 0
    total_accuracy = 0

    with torch.no_grad():
        for image, mask in tqdm(data, desc="Evaluating"):
            image, mask = image.to(device), mask.to(device)

            outputs = net(image)

            mask = mask.squeeze(1).long()

            loss = criterion(outputs, mask)
            dice = utils.dice_score(outputs, mask)
            accuracy = utils.accuracy_score(outputs, mask)

            total_loss += loss.item()
            total_dice += dice.item()
            total_accuracy += accuracy.item()

    avg_loss = total_loss / len(data)
    avg_dice = total_dice / len(data)
    avg_accuracy = total_accuracy / len(data)

    return avg_loss, avg_dice, avg_accuracy
```

直接 call 寫在 utils 的函數算準確率。



```python
def dice_score(pred_mask, gt_mask):
    pred_mask = torch.argmax(pred_mask, dim=1)

    gt_mask = gt_mask.squeeze(1) # [batch_size, height, width]

    pred_mask = pred_mask.float()
    gt_mask = gt_mask.float()

    intersection = (pred_mask * gt_mask).sum(dim=(1, 2))
    union = pred_mask.sum(dim=(1, 2)) + gt_mask.sum(dim=(1, 2))

    dice = 2. * intersection / union
    return dice.mean()

def accuracy_score(pred, target):
    pred = torch.argmax(pred, dim=1)
    correct = (pred == target).float()
    return correct.sum() / correct.numel()
```

inference.py

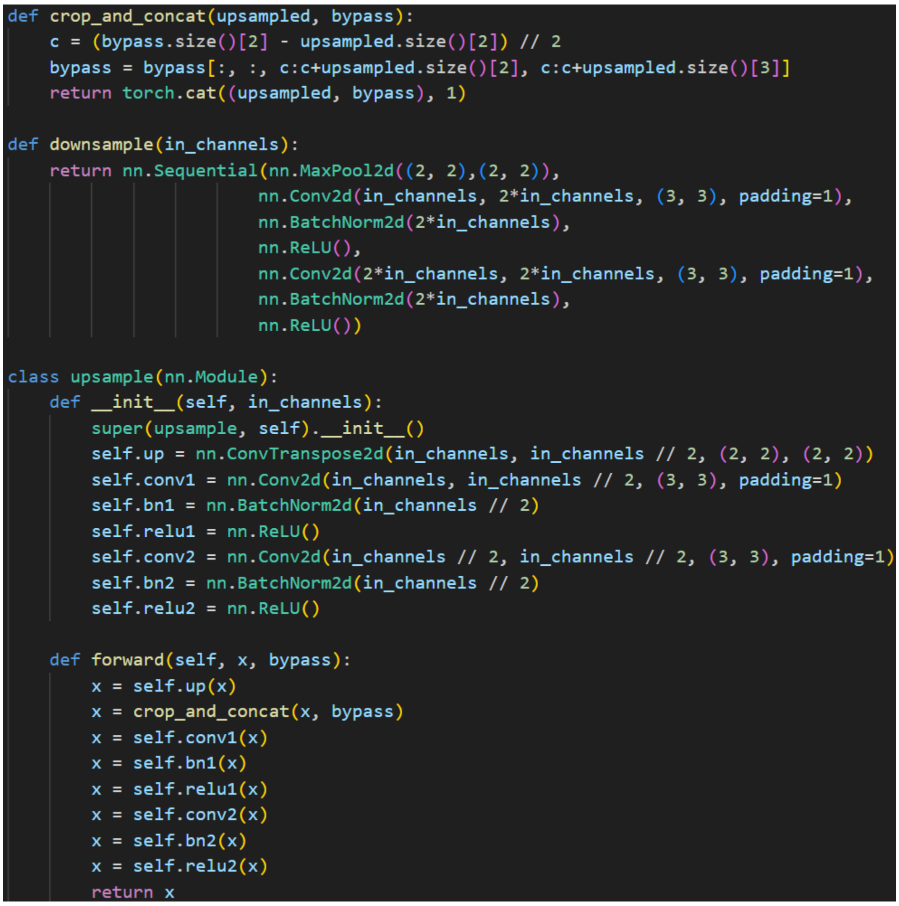


```python
def get_args():
    parser = argparse.ArgumentParser(description='Predict masks from input images')
    parser.add_argument('--model', default='MODEL.pth', help='path to the stored model weight')
    parser.add_argument('--data_path', type=str, help='path to the input data')
    parser.add_argument('--batch_size', '-b', type=int, default=1, help='batch size')

    return parser.parse_args()

if __name__ == '__main__':
    args = get_args()

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    if args.model == './Lab3/saved_models/UNet.pth':
        model = unet.UNet(3, 2).to(device)
    else:
        model = resnet34_unet.ResNet34_UNet().to(device)

    model.load_state_dict(torch.load(args.model, map_location=device))
    test_loader = oxford_pet.load_dataset(args.data_path, "test", args.batch_size)
    loss_score, dice_score, accuracy_score = evaluate.evaluate(model, test_loader, device)
    print(f"Test Loss: {loss_score:.4f}")
    print(f"Test Dice Score: {dice_score:.4f}")
    print(f"Test Accuracy: {accuracy_score:.4f}")
```

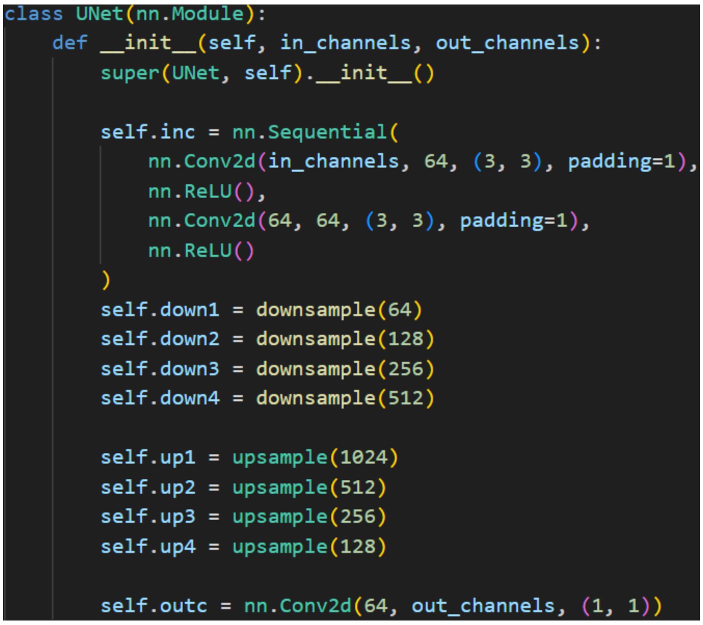
用 evaluate 裡寫好的 function 直接運算 Dice Score, Loss, Accuracy，準確率計算為預測相同的 pixel 數除以總 pixel 數，有一點值得注意的是只有 Training Set 需要做 preprocessing，其他 Data Set 做預處理會有 train test contamination 的問題。

- ➢ Details of your model

UNet



```
def crop_and_concat(upsampled, bypass):
    c = (bypass.size()[2] - upsampled.size()[2]) // 2
    bypass = bypass[:, :, c:c+upsampled.size()[2], c:c+upsampled.size()[3]]
    return torch.cat((upsampled, bypass), 1)

def downsample(in_channels):
    return nn.Sequential(nn.MaxPool2d((2, 2),(2, 2)),
                        nn.Conv2d(in_channels, 2*in_channels, (3, 3), padding=1),
                        nn.BatchNorm2d(2*in_channels),
                        nn.ReLU(),
                        nn.Conv2d(2*in_channels, 2*in_channels, (3, 3), padding=1),
                        nn.BatchNorm2d(2*in_channels),
                        nn.ReLU())

class upsample(nn.Module):
    def __init__(self, in_channels):
        super(upsample, self).__init__()
        self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, (2, 2), (2, 2))
        self.conv1 = nn.Conv2d(in_channels, in_channels // 2, (3, 3), padding=1)
        self.bn1 = nn.BatchNorm2d(in_channels // 2)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels // 2, in_channels // 2, (3, 3), padding=1)
        self.bn2 = nn.BatchNorm2d(in_channels // 2)
        self.relu2 = nn.ReLU()

    def forward(self, x, bypass):
        x = self.up(x)
        x = crop_and_concat(x, bypass)
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu2(x)
        return x
```



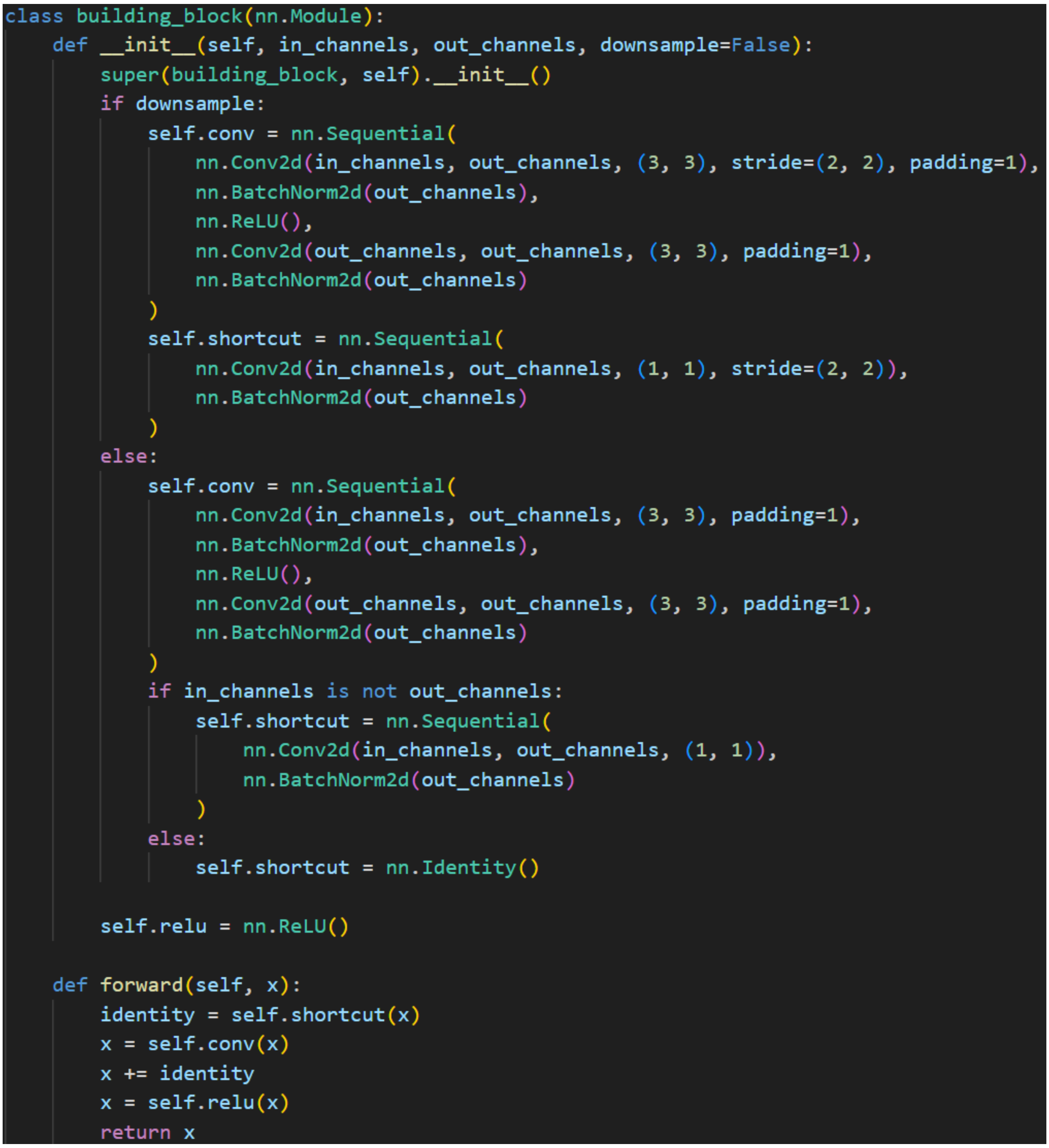
```
class UNet(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UNet, self).__init__()

        self.inc = nn.Sequential(
            nn.Conv2d(in_channels, 64, (3, 3), padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 64, (3, 3), padding=1),
            nn.ReLU()
        )
        self.down1 = downsample(64)
        self.down2 = downsample(128)
        self.down3 = downsample(256)
        self.down4 = downsample(512)

        self.up1 = upsample(1024)
        self.up2 = upsample(512)
        self.up3 = upsample(256)
        self.up4 = upsample(128)

        self.outc = nn.Conv2d(64, out_channels, (1, 1))
```

先實作出 downsample、upsample Module 跟 crop_and_concat 函數。

downsample 由 maxpoll 跟兩個 3x3 conv 構成，upsample 做轉置捲積減少 channel，跟前面的 downsample 的輸出 concat 再做兩次捲積，捲積完會做 BN 跟 ReLU 。

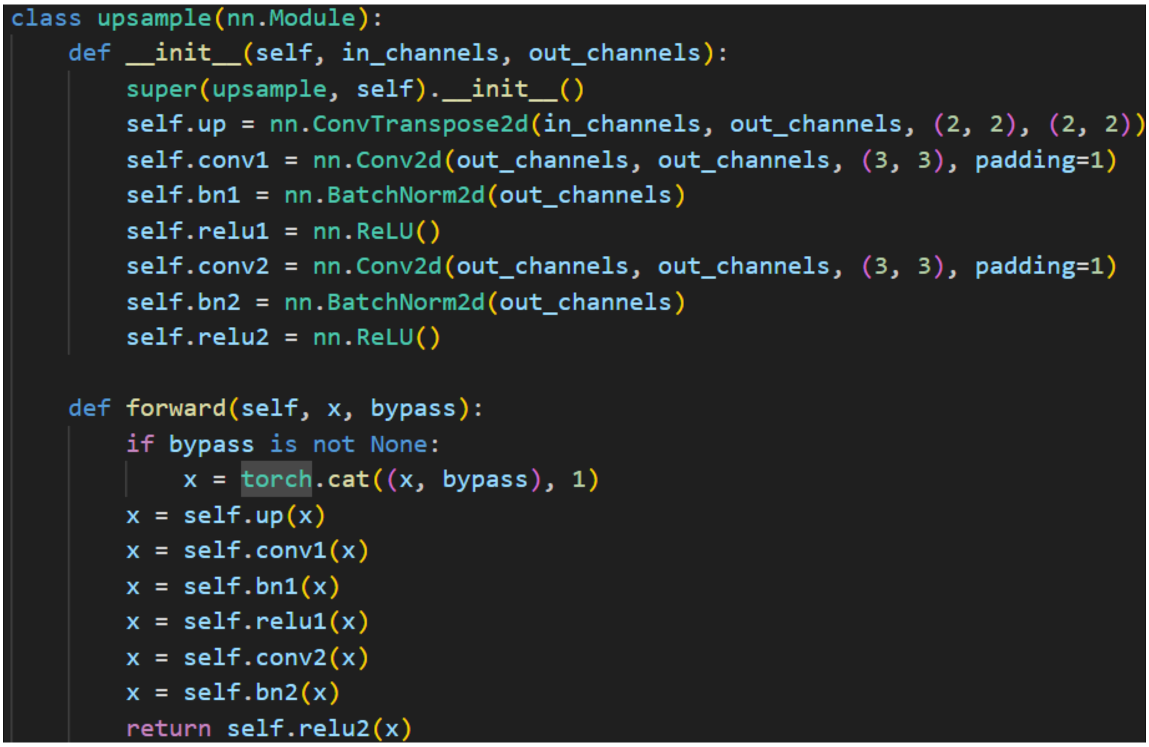
ResNet34_UNet

先依照 paper 實做殘差網路的基本 module。



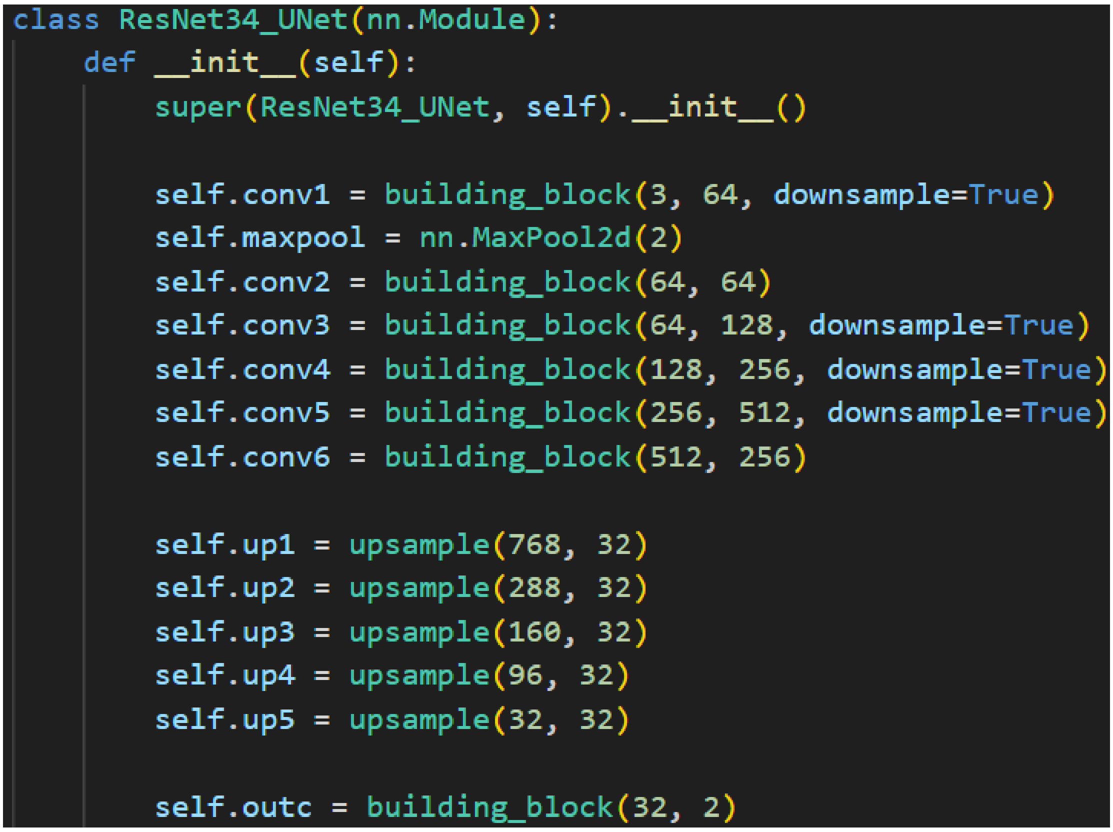
```python
class building_block(nn.Module):
    def __init__(self, in_channels, out_channels, downsample=False):
        super(building_block, self).__init__()
        if downsample:
            self.conv = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, (3, 3), stride=(2, 2), padding=1),
                nn.BatchNorm2d(out_channels),
                nn.ReLU(),
                nn.Conv2d(out_channels, out_channels, (3, 3), padding=1),
                nn.BatchNorm2d(out_channels)
            )
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, (1, 1), stride=(2, 2)),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, (3, 3), padding=1),
                nn.BatchNorm2d(out_channels),
                nn.ReLU(),
                nn.Conv2d(out_channels, out_channels, (3, 3), padding=1),
                nn.BatchNorm2d(out_channels)
            )
            if in_channels is not out_channels:
                self.shortcut = nn.Sequential(
                    nn.Conv2d(in_channels, out_channels, (1, 1)),
                    nn.BatchNorm2d(out_channels)
                )
            else:
                self.shortcut = nn.Identity()

        self.relu = nn.ReLU()

    def forward(self, x):
        identity = self.shortcut(x)
        x = self.conv(x)
        x += identity
        x = self.relu(x)
        return x
```

building block 在 downsample 時會做 stride=2 讓高寬減半，而當 channel 數不一樣時候使用 paper 中(b)option，做(1, 1)conv 增加讓模型學習。



```
class upsample(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(upsample, self).__init__()
        self.up = nn.ConvTranspose2d(in_channels, out_channels, (2, 2), (2, 2))
        self.conv1 = nn.Conv2d(out_channels, out_channels, (3, 3), padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, (3, 3), padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.relu2 = nn.ReLU()

    def forward(self, x, bypass):
        if bypass is not None:
            x = torch.cat((x, bypass), 1)
        x = self.up(x)
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        return self.relu2(x)
```

upsample 跟 UNet 差不多，只是調整 channel size 跟 concat 的順序以符合模型。



```
class ResNet34_UNet(nn.Module):
    def __init__(self):
        super(ResNet34_UNet, self).__init__()

        self.conv1 = building_block(3, 64, downsample=True)
        self.maxpool = nn.MaxPool2d(2)
        self.conv2 = building_block(64, 64)
        self.conv3 = building_block(64, 128, downsample=True)
        self.conv4 = building_block(128, 256, downsample=True)
        self.conv5 = building_block(256, 512, downsample=True)
        self.conv6 = building_block(512, 256)

        self.up1 = upsample(768, 32)
        self.up2 = upsample(288, 32)
        self.up3 = upsample(160, 32)
        self.up4 = upsample(96, 32)
        self.up5 = upsample(32, 32)

        self.outc = building_block(32, 2)
```

3.Data Preprocessing (20%)

- How you preprocessed your data?

  將照片轉換成固定 256x256 像素，image 使用 BILINEAR，mask 使用 NEAREST，並將圖像從 HWC 格式轉為 CHW， data augmentation 使用了隨機旋轉跟水平翻轉。

- What makes your method unique?

  我使用的方法並不涉及調整 pixel 的數值，目的是希望模型訓練可以模擬現實人眼看到的畫面，把圖片旋轉、旋轉翻轉對於人來說是幾乎沒影響的，所以希望機器可以用人眼的視覺學習這一任務。

4.Analyze on the experiment results (20%)

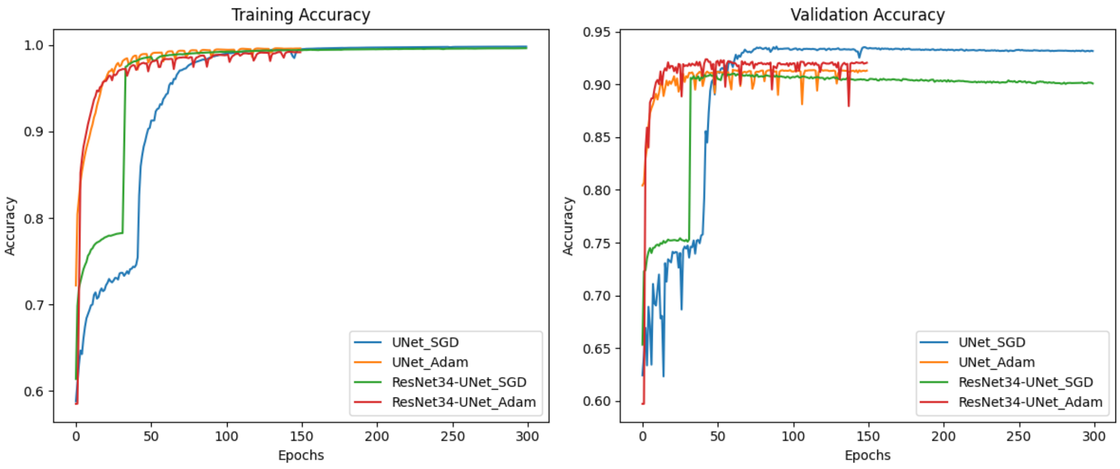
- What did you explore during the training process?

  我的顯卡型號是 3060TI 記憶體 32GB，訓練模型時最佳的 batch size 為 8，在 batch size 16 的時候會跑得比較久，而 32 會直接超出記憶體。

  LR 若大於 1e-2 模型會無法收斂，驗證準確率卡在 0.90 上下，Dice 到不了 90%。

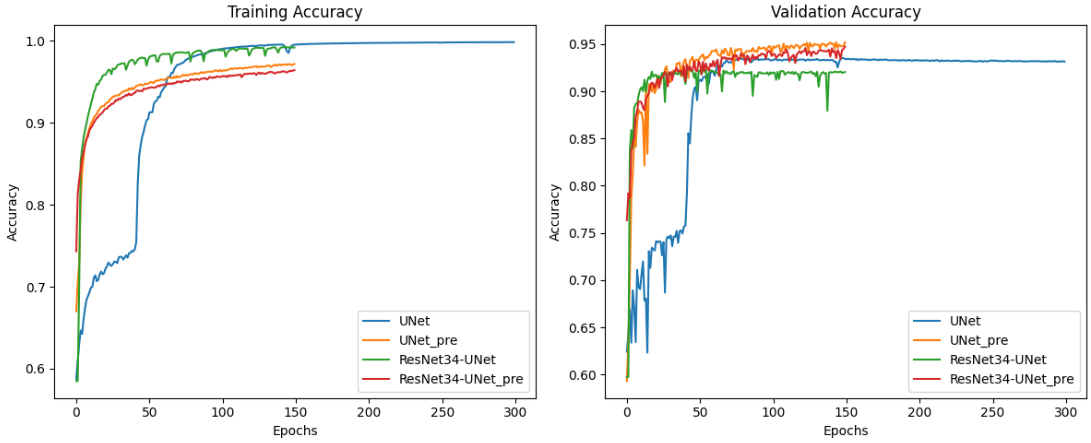
  一開始的訓練都沒有將圖片做旋轉翻轉，先找出最佳訓練參數，最後再加入 preprocessing 進行比較。

  經過訓練發現最終訓練準確率幾乎相同，最終都可以到 0.99，但學習曲線跟驗證準確率有很大的差別。我做的 UNet 在使用 SGD 上相比 Adam 有比較好的驗證準確率，而 ResNet34_UNet 則相反。

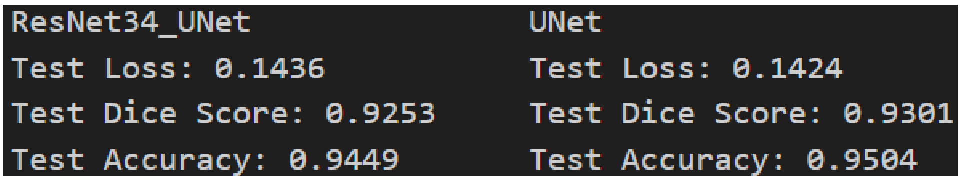


驗證準確率由大到小排為：UNet_SGD > ResNet34_UNet_Adam > UNet_Adam > ResNet34_UNet_SGD

在學習曲線上，可以明顯看到 SGD 在 epoch30 時有明顯的瓶頸，之後準確率會大幅提升。主要是 Learning rate decay 讓模型快速收斂。



最後加入 preprocessing 跟兩模型的最佳解做比較，可以看到加入隨機翻轉後模型收斂變快，訓練準確率降低了，但換來了更好的驗證準確率，代表預處理讓模型有更好的泛化能力。

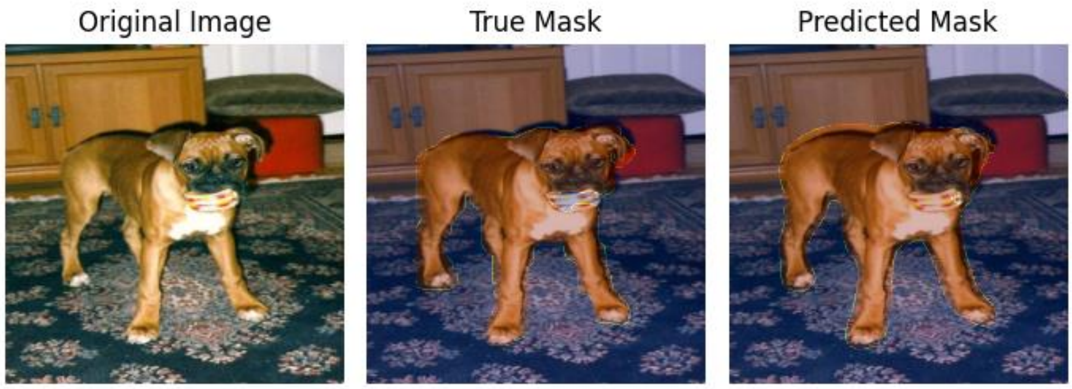


```
ResNet34_UNet                  UNet
Test Loss: 0.1436              Test Loss: 0.1424
Test Dice Score: 0.9253        Test Dice Score: 0.9301
Test Accuracy: 0.9449          Test Accuracy: 0.9504
```

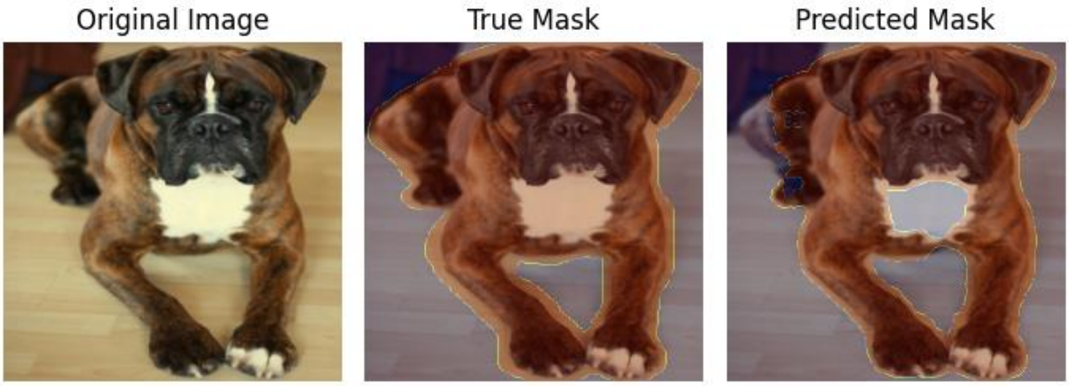
最佳模型測試結果

➢ Found any characteristics of the data?

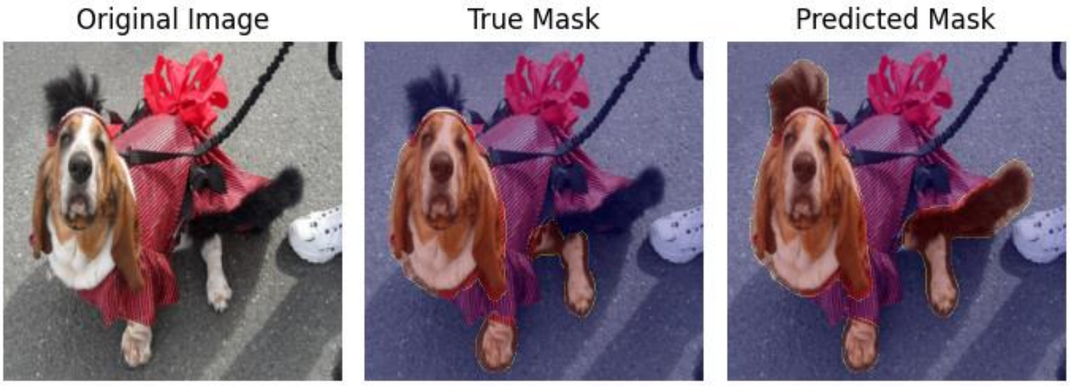
若給的圖片是單色的背景加上中間主體，基本預測十分準確，而背景複雜但有規律也很準確。



但若主題跟背景顏色相近，或身上有穿項圈或是貓咪的鬍鬚，這些都容易讓模型誤判。認為是在預處理 resize 就已經損失了大量細節，而在做捲積、Pooling 時又繼續丟失特徵，或是發生平移，沒了圖片的相對位置，模型底層只能學到很粗略的特徵，所以容易誤判或是沒辦法做很細緻預測。

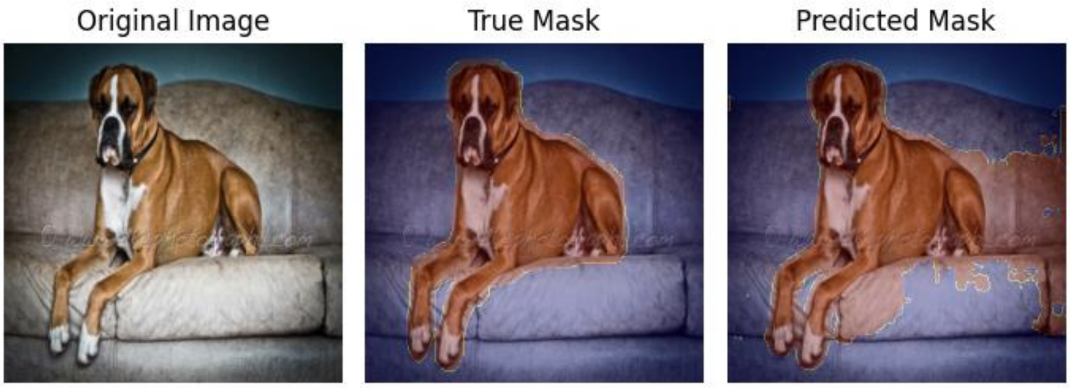


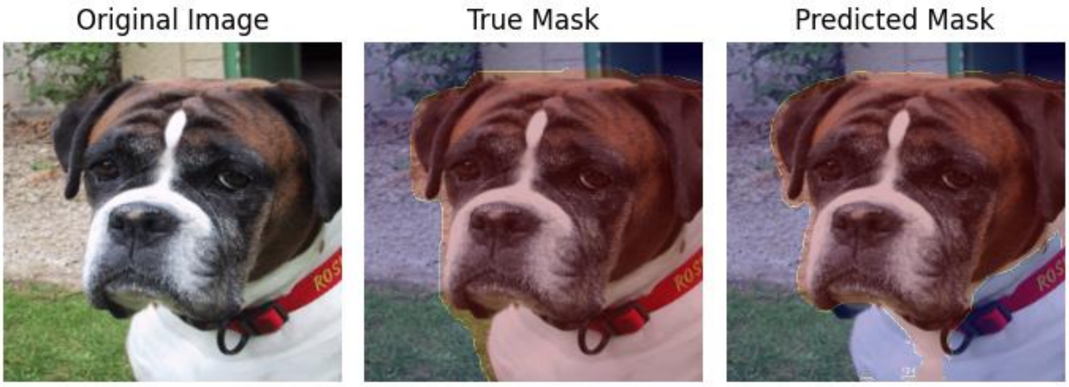
中間白色部分被視為背景，而衣服絨毛部分也被一起考慮



這些結果表示模型現階段是能完美完成簡單圖片的二元分割任務，但對於動物以外的東西沒有認知，不過這也吻合當初對這任務的描述。

有觀察到模型把某些既定印象學得太強，例如前景的形狀，有些圖片明明不複雜但會有很怪的預測，可能需要增加 dropout 讓 model 不要 over-fit。





GroundTruth 的品質也很重要，很多張照片都會把項圈當背景但是上面這張卻沒有，鬍鬚那些部分也沒有達到 pixel 級別的準確度，這些都會導致模型預測結果不好。

5.Execution command (0%)

- The command and parameters for the training process

若是要訓練 UNet，請執行 ：

```
python ./Lab3/src/train.py --data_path ./Lab3/dataset/oxford-iiit-pet/ --epochs 150 --batch_size 8 --learning-rate 1e-2 --model UNet
```

若是要訓練 ResNet34_UNet，請執行 ：

```
python ./Lab3/src/train.py --data_path ./Lab3/dataset/oxford-iiit-pet/ --epochs 150 --batch_size 8 --learning-rate 1e-2 --model ResNet34_UNet
```

- The command and parameters for the inference process

若是要測試 UNet，請執行 ：

```
python ./Lab3/src/inference.py --data_path ./Lab3/dataset/oxford-iiit-pet/ --batch_size 8 --model ./Lab3/saved_models/UNet.pth
```

若是要測試 ResNet34_UNet，請執行 ：

```
python ./Lab3/src/inference.py --data_path ./Lab3/dataset/oxford-iiit-pet/ --batch_size 8 --model ./Lab3/saved_models/ResNet34_UNet.pth
```

6. Discussion (20%)

- What architecture may bring better results?

模型會學過頭並且認不得動物身上的衣服跟細節，所以要加入 dropout 且將模型變得複雜，盡量減少資訊損失，可以加入 Attention 機制，將注意力

分別去辨識不同物件以達到更高準確率。

- What are the potential research topics in this task?

  1. 想幫家裡寵物做 Line 貼圖但懶得自己去背的時候。

  2. Tesla 自動駕駛，電腦視覺會判斷物體的外型、輪廓。

  3. 用在醫學影像做初步診斷，可以識別腫瘤區塊，減輕醫生的負擔、提升手術成功機率。