DLP_LAB3

— 313551133 陳軒宇

1. Overview of your lab 3 (10%)

本次實驗的目的在於探索深度學習在二元語義分割 (Binary Semantic Segmentation) 任務中的應用，我們需要實現並比較 UNet 和 ResNet34+UNet 這兩種不同的神經網絡架構，並使用 Oxford-IIIT Pet Dataset 來訓練和測試模型，以評估它們在寵物圖像分割中的性能。

2. Implementation Details (30%)

A. Details of your training, evaluating, inferencing code

這三個部分都能透過命令行參數來靈活配置運行參數，如資料路徑、批次大小、模型類型等。

訓練部分（`train.py`）

- 使用 `DataLoader` 來批量加載和處理數據。
- 模型可以選擇 `UNet` 或 `ResNet34_UNet`。
- 使用 `CrossEntropyLoss` 作為損失函數、`Adam` 作為優化器。
- 使用了 `ExponentialLR` 學習率調度器，以動態調整學習率。
- 在訓練過程中，除了評估在 training set 上的 loss 外，我們也同時在 validation set 上透過 loss 和 Dice score 評估模型性能。
- 由於訓練時間過長，實現了檢查點保存和加載機制，允許從中斷處繼續訓練。
- 每隔 25 個 epoch 保存一次模型，並根據在 validation set 上的 Dice score 和 loss 保存了最佳 loss 和 Dice score 對應的模型（雖然沒有用到）。

```
def train(args):
    # implement the training function here

    assert args.model in ["unet", "resnet34_unet"]

    print(device)

    train_dataset = load_dataset(args.data_path, "train")
    train_loader = DataLoader(train_dataset, batch_size=args.batch_size,
shuffle=True)
    valid_dataset = load_dataset(args.data_path, "valid")
    valid_loader = DataLoader(valid_dataset, batch_size=args.batch_size,
shuffle=False)

    if args.model == 'unet':
        model = UNet(input_channels=3, n_classes=1).to(device)
    else:
        model = ResNet34_UNet(in_channels=3, n_classes=1).to(device)

    # Set the save path
    save_path = os.path.join(args.save_path, args.model)
    model_path = os.path.join(save_path, args.model + '.pth')
    ckpt_path = os.path.join(save_path, args.model + '.ckpt')
    final_model_path = os.path.join(save_path, args.model + '_final.pth')

    if not os.path.exists(args.save_path):
        os.makedirs(args.save_path)
    if not os.path.exists(save_path):
        os.makedirs(save_path)

    print(f"Model saved at {save_path}")

    # Set the criterion, optimizer and scheduler
    critirion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)
    scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, 0.99)
```

```
    # keep track of performance metrics and some other information
    train_losses = []
    valid_losses = []
    valid_scores = []
    start_epoch = 1
    best_valid_loss = np.inf
    best_valid_score = 0.0

    # load the checkpoint if it exists
    if os.path.exists(ckpt_path):
        ckpt = torch.load(ckpt_path)
        model.load_state_dict(ckpt['model_state_dict'])
        optimizer.load_state_dict(ckpt['optimizer_state_dict'])
        scheduler.load_state_dict(ckpt['scheduler_state_dict'])
        start_epoch = ckpt['epoch'] + 1
        train_losses = ckpt['train_losses']
        valid_losses = ckpt['valid_losses']
        valid_scores = ckpt['valid_scores']
        best_valid_loss = ckpt['best_valid_loss']
        best_valid_score = ckpt['best_valid_score']
        print(f"Model loaded from {ckpt_path}")

    # Training loop
    for epoch in range(start_epoch, args.epochs + 1):
        # train the model
        model.train() # set the model to training mode
        train_loss = 0

        for batch in tqdm(train_loader, desc=f"Epoch {epoch}/{args.epochs} -
Train"):
            images, masks = batch["image"].to(device), batch["mask"].to(device)
            outputs = model(images)

            # flatten the output and mask tensors for cross entropy loss
            # loss = critirion(outputs, masks)
            loss = critirion(outputs.flatten(start_dim=1),
masks.flatten(start_dim=1))
            train_loss += loss.item() * images.size(0)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        train_loss /= len(train_loader.dataset)

        # evaluate the model
        model.eval() # set the model to evaluation mode
        valid_loss = 0.0
        valid_score = 0.0 # dice score

        with torch.no_grad(): # without tracking history
            for batch in tqdm(valid_loader, desc=f"Epoch {epoch}/{args.epochs}
- Valid"):
                images, masks = batch["image"].to(device),
batch["mask"].to(device)

                outputs = model(images)
                # flatten the output and mask tensors for cross entropy loss
```

```
                # loss = critirion(outputs, masks)
                loss = critirion(outputs.flatten(start_dim=1),
masks.flatten(start_dim=1))
                valid_loss += loss.item() * images.size(0)
                valid_score += dice_score(outputs, masks) * images.size(0)

        valid_loss /= len(valid_loader.dataset)
        valid_score /= len(valid_loader.dataset)

        train_losses.append(train_loss)
        valid_losses.append(valid_loss)
        valid_scores.append(valid_score)

        print(f"Epoch {epoch}/{args.epochs}, Train Loss: {train_loss:.4f}, Val
Loss: {valid_loss:.4f}, Val Score: {valid_score:.4f}, LR:
{scheduler.get_last_lr()[0]:.6f}")

        scheduler.step() # update the learning rate

        if valid_loss < best_valid_loss:
            best_valid_loss = valid_loss
            torch.save(model.state_dict(), f"
{save_path}/{args.model}_best_loss.pth")

        if valid_score > best_valid_score:
            best_valid_score = valid_score
            torch.save(model.state_dict(), f"
{save_path}/{args.model}_best_score.pth")

        if epoch % 25 == 0:
            torch.save(model.state_dict(), f"
{save_path}/{args.model}_epoch_{epoch}.pth")
            info = {"model": model, "train_losses": train_losses,
"valid_losses": valid_losses, "valid_scores": valid_scores}
            with open(f"{save_path}/{args.model}_epoch_{epoch}.pkl", "wb") as
f: # save the loss history
                pickle.dump(info, f)

        # save checkpoint
        torch.save(
            {
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'scheduler_state_dict': scheduler.state_dict(),
                'epoch': epoch,
                'train_losses': train_losses,
                'valid_losses': valid_losses,
                'valid_scores': valid_scores,
                'best_valid_loss': best_valid_loss,
                'best_valid_score': best_valid_score
            }, ckpt_path)

    torch.save(model.state_dict(), final_model_path)
```

命令行參數部分，用於訓練模型：

```
def get_args():
    parser = argparse.ArgumentParser(description='Train the UNet on images and
target masks')
    parser.add_argument('--data_path', type=str, default='dataset', help='path
of the input data')
    parser.add_argument('--epochs', '-e', type=int, default=500, help='number
of epochs')
    parser.add_argument('--batch_size', '-b', type=int, default=8, help='batch
size')
    parser.add_argument('--learning-rate', '-lr', type=float, default=1e-5,
help='learning rate')
    parser.add_argument("--model", type=str, default="unet", help="model to use
(unet/resnet34_unet)")
    parser.add_argument("--save_path", type=str, default="./saved_models/",
help="path to save the model")
    return parser.parse_args()

if __name__ == "__main__":
    args = get_args()
    train(args)
```

評估部分（`evaluate.py`）

- 實現了 `evaluate` 函數，用來在 test set 上評估模型性能。
- 使用 Dice score 作為主要評估指標，但由於我在 `utils.py` 中定義的 `dice_score()` 函數返回的是平均值，所以在累加時先將結果乘以 $batch\_size$ ，最後再除以整個測試集的數量，以確保得到的結果是所有圖片的 Dice score 的平均值。
- 支持批量處理以提高評估效率。

函數部分：

```
def evaluate(model, data, device,desc=""):
    # implement the evaluation function here
    model.eval()
    scores = 0
    with torch.no_grad():
        for batch in tqdm(data, desc=desc):
            images, masks = batch["image"].to(device), batch["mask"].to(device)
            outputs = model(images)
            scores += dice_score(outputs, masks) * images.size(0)
    return scores / len(data.dataset)
```

命令行參數部分，用於評估模型在 test set 上的性能：

```
def get_args():
    parser = argparse.ArgumentParser(description='Evaluate the UNet and
ResNet34-UNet on images and target masks')
    parser.add_argument('--data_path', type=str, default='dataset', help='path
of the input data')
    parser.add_argument('--batch_size', '-b', type=int, default=8, help='batch
size')
```

```
    parser.add_argument("--model", type=str, default="unet", help="model to use
(unet/resnet34_unet)")
    parser.add_argument("--model_path", type=str,
default="./saved_models/unet/unet_final.pth", help="path to load the model")
    return parser.parse_args()

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

    args.model = "unet" if args.model.lower().startswith("u") else
"resnet34_unet"
    assert args.model in ["unet", "resnet34_unet"]

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

    test_dataset = load_dataset(args.data_path, "test")
    test_loader = DataLoader(test_dataset, batch_size=args.batch_size,
shuffle=False)

    if args.model == 'unet':
        model = UNet(input_channels=3, n_classes=1).to(device)
    else:
        model = ResNet34_UNet(in_channels=3, n_classes=1).to(device)

    if not os.path.exists(args.model_path):
        raise FileNotFoundError(f"Model not found at {args.model_path}")

    model.load_state_dict(torch.load(args.model_path))
    model.eval()

    scores = evaluate(model, test_loader, device, desc=f"
{os.path.basename(args.model_path)}")
    print(f"Test Dice Score: {scores:.4f}")
```

推論部分（`inference.py`）

- 實現了兩種推論模式：批量推論和單張圖片推論。
- `predict` 函數用於批量處理整個資料集。
- `predict_single` 函數用於處理單張圖片。
- 實現了結果可視化功能，可以將預測遮罩與原圖和真實遮罩並排顯示，後續 4.B 中會展示。
- 支持將推論結果保存為圖片文件。

函數部分：

```
def predict(model, data, device, desc="Predicting"):
    model.eval()
    images = []
    preds_masks = []
    gt_masks = []
    with torch.no_grad():
        for batch in tqdm(data, desc=desc):
            inputs = batch["image"].to(device)
            outputs = model(inputs)
```

```
            images.append(inputs.cpu().numpy())
            preds_masks.append(outputs.cpu().numpy())
            gt_masks.append(batch["mask"].cpu().numpy())
    return dict(images=np.concatenate(images, axis=0),
preds_masks=np.concatenate(preds_masks, axis=0),
gt_masks=np.concatenate(gt_masks, axis=0))

def predict_single(model, data, device):
    model.eval()
    with torch.no_grad():
        image = torch.from_numpy(data["image"])
        image = image.unsqueeze_(0).to(device)
        pred_mask = model(image).squeeze(0) # (1, 1, H, W) -> (1, H, W)
    return dict(image=data["image"], pred_mask=pred_mask.cpu().numpy(),
gt_mask=data["mask"])

def save_images(images, pred_masks, gt_masks, filenames, output_path,
alpha=0.5):
    if not os.path.exists(output_path):
        os.makedirs(output_path)
    for (image, pred_mask, gt_mask, filename) in tqdm(zip(images, pred_masks,
gt_masks, filenames), total=len(images), desc="Saving images"):
        image = (image * 255).astype(np.uint8)
        pred_mask = (pred_mask > 0.5).astype(np.uint8) * 255 # 二值化
        gt_mask = (gt_mask > 0.5).astype(np.uint8) * 255 # 二值化

        image = image.transpose(1, 2, 0)
        pred_mask = pred_mask.transpose(1, 2, 0)
        gt_mask = gt_mask.transpose(1, 2, 0)

        pred_mask = np.repeat(pred_mask, 3, axis=2)
        gt_mask = np.repeat(gt_mask, 3, axis=2)

        image = Image.fromarray(image)
        pred_mask = Image.fromarray(pred_mask)
        gt_mask = Image.fromarray(gt_mask)
        blend1 = Image.blend(image, pred_mask, alpha=alpha)
        blend2 = Image.blend(image, gt_mask, alpha=alpha)

        blend1.save(os.path.join(output_path, f"{filename}_pred.png"))
        blend2.save(os.path.join(output_path, f"{filename}_gt.png"))

def show_single_result(image, pred_mask, gt_mask, suptitle, alpha=0.5,
save_path=None):
    image = (image * 255).astype(np.uint8)
    pred_mask = (pred_mask > 0.5).astype(np.uint8) * 255 # 二值化
    gt_mask = (gt_mask > 0.5).astype(np.uint8) * 255 # 二值化

    pred_mask = np.repeat(pred_mask, 3, axis=0)
    gt_mask = np.repeat(gt_mask, 3, axis=0)

    blend1 = cv2.addWeighted(image, alpha, pred_mask, 1 - alpha, 0)
    blend2 = cv2.addWeighted(image, alpha, gt_mask, 1 - alpha, 0)

    data = dict(image=image, pred_mask=blend1, gt_mask=blend2)
    show_dataset_sample(data, suptitle, save_path)
```

命令行參數部分：

```
def get_args():

    parser = argparse.ArgumentParser(description='Predict masks from input
images')
    parser.add_argument('--data_path', type=str, default='dataset', help='path
of the input data')
    parser.add_argument('--batch_size', '-b', type=int, default=4, help='batch
size')
    parser.add_argument("--model", type=str, default="unet", help="model to use
(unet/resnet34_unet)")
    parser.add_argument("--model_path", type=str,
default="./saved_models/unet/unet_final.pth", help="path to load the model")
    parser.add_argument("--output_path", type=str, default="output", help="path
to save the output")
    parser.add_argument('--idx', '-i', type=int, default=-1, help='index of the
image to predict')
    return parser.parse_args()

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

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

    args.model = "unet" if args.model.lower().startswith("u") else
"resnet34_unet"
    assert args.model in ["unet", "resnet34_unet"]

    if not os.path.exists(args.output_path):
        os.makedirs(args.output_path)

    if args.model == 'unet':
        model = UNet(input_channels=3, n_classes=1).to(device)
    else:
        model = ResNet34_UNet(in_channels=3, n_classes=1).to(device)

    if not os.path.exists(args.model_path):
        raise FileNotFoundError(f"Model not found at {args.model_path}")

    model.load_state_dict(torch.load(args.model_path))
    model.eval()

    test_dataset = load_dataset(args.data_path, "test")
    if args.idx == -1: # 預測所有資料
        test_loader = DataLoader(test_dataset, batch_size=args.batch_size,
shuffle=False)
        data = predict(model, test_loader, device, desc="Predicting")
        save_images(data["images"], data["preds_masks"], data["gt_masks"],
test_dataset.filenames, os.path.join(args.output_path, f"{args.model}"))
    else:
        data = predict_single(model, test_dataset[args.idx], device)
        dice_score = dice_score(data["pred_mask"], data["gt_mask"])
        print(f"Dice Score of test[{args.idx}]: {dice_score:.4f}")
        suptitle = f"{args.model} - test[{args.idx}]:
{test_dataset.filenames[args.idx]} - Dice Score: {dice_score:.4f}"
        show_single_result(data["image"], data["pred_mask"], data["gt_mask"],
```

```
suptitle, save_path=os.path.join(args.output_path, f"
{args.model}_{args.idx}.png"))
```

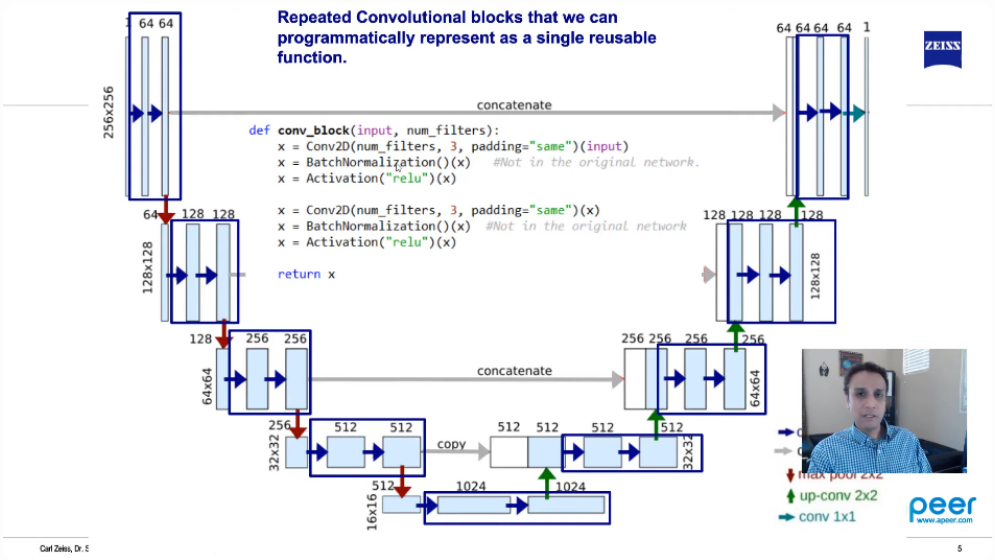
B. Details of your model (UNet & ResNet34_UNet)

在本次 Lab 中，我們實現了兩種不同的模型架構：`UNet` 和 `ResNet34_UNet`。

UNet

```
----------------------------------------------------------------
        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
         ConvBlock-7         [-1, 64, 256, 256]               0
         MaxPool2d-8         [-1, 64, 128, 128]               0
      EncoderBlock-9  [[-1, 64, 256, 256], [-1, 64, 128, 128]]               0
           Conv2d-10        [-1, 128, 128, 128]          73,856
      BatchNorm2d-11        [-1, 128, 128, 128]             256
             ReLU-12        [-1, 128, 128, 128]               0
           Conv2d-13        [-1, 128, 128, 128]         147,584
      BatchNorm2d-14        [-1, 128, 128, 128]             256
             ReLU-15        [-1, 128, 128, 128]               0
        ConvBlock-16        [-1, 128, 128, 128]               0
        MaxPool2d-17          [-1, 128, 64, 64]               0
     EncoderBlock-18  [[-1, 128, 128, 128], [-1, 128, 64, 64]]               0
           Conv2d-19          [-1, 256, 64, 64]         295,168
      BatchNorm2d-20          [-1, 256, 64, 64]             512
             ReLU-21          [-1, 256, 64, 64]               0
           Conv2d-22          [-1, 256, 64, 64]         590,080
      BatchNorm2d-23          [-1, 256, 64, 64]             512
             ReLU-24          [-1, 256, 64, 64]               0
        ConvBlock-25          [-1, 256, 64, 64]               0
        MaxPool2d-26          [-1, 256, 32, 32]               0
     EncoderBlock-27  [[-1, 256, 64, 64], [-1, 256, 32, 32]]               0
           Conv2d-28          [-1, 512, 32, 32]       1,180,160
      BatchNorm2d-29          [-1, 512, 32, 32]           1,024
             ReLU-30          [-1, 512, 32, 32]               0
           Conv2d-31          [-1, 512, 32, 32]       2,359,808
      BatchNorm2d-32          [-1, 512, 32, 32]           1,024
             ReLU-33          [-1, 512, 32, 32]               0
        ConvBlock-34          [-1, 512, 32, 32]               0
        MaxPool2d-35          [-1, 512, 16, 16]               0
     EncoderBlock-36  [[-1, 512, 32, 32], [-1, 512, 16, 16]]               0
           Conv2d-37         [-1, 1024, 16, 16]       4,719,616
      BatchNorm2d-38         [-1, 1024, 16, 16]           2,048
             ReLU-39         [-1, 1024, 16, 16]               0
           Conv2d-40         [-1, 1024, 16, 16]       9,438,208
      BatchNorm2d-41         [-1, 1024, 16, 16]           2,048
             ReLU-42         [-1, 1024, 16, 16]               0
        ConvBlock-43         [-1, 1024, 16, 16]               0
  ConvTranspose2d-44          [-1, 512, 32, 32]       2,097,664
```

```
           Conv2d-45          [-1, 512, 32, 32]       4,719,104
      BatchNorm2d-46          [-1, 512, 32, 32]           1,024
             ReLU-47          [-1, 512, 32, 32]               0
           Conv2d-48          [-1, 512, 32, 32]       2,359,808
      BatchNorm2d-49          [-1, 512, 32, 32]           1,024
             ReLU-50          [-1, 512, 32, 32]               0
        ConvBlock-51          [-1, 512, 32, 32]               0
     DecoderBlock-52          [-1, 512, 32, 32]               0
  ConvTranspose2d-53          [-1, 256, 64, 64]         524,544
           Conv2d-54          [-1, 256, 64, 64]       1,179,904
      BatchNorm2d-55          [-1, 256, 64, 64]             512
             ReLU-56          [-1, 256, 64, 64]               0
           Conv2d-57          [-1, 256, 64, 64]         590,080
      BatchNorm2d-58          [-1, 256, 64, 64]             512
             ReLU-59          [-1, 256, 64, 64]               0
        ConvBlock-60          [-1, 256, 64, 64]               0
     DecoderBlock-61          [-1, 256, 64, 64]               0
  ConvTranspose2d-62        [-1, 128, 128, 128]         131,200
           Conv2d-63        [-1, 128, 128, 128]         295,040
      BatchNorm2d-64        [-1, 128, 128, 128]             256
             ReLU-65        [-1, 128, 128, 128]               0
           Conv2d-66        [-1, 128, 128, 128]         147,584
      BatchNorm2d-67        [-1, 128, 128, 128]             256
             ReLU-68        [-1, 128, 128, 128]               0
        ConvBlock-69        [-1, 128, 128, 128]               0
     DecoderBlock-70        [-1, 128, 128, 128]               0
  ConvTranspose2d-71         [-1, 64, 256, 256]          32,832
           Conv2d-72         [-1, 64, 256, 256]          73,792
      BatchNorm2d-73         [-1, 64, 256, 256]             128
             ReLU-74         [-1, 64, 256, 256]               0
           Conv2d-75         [-1, 64, 256, 256]          36,928
      BatchNorm2d-76         [-1, 64, 256, 256]             128
             ReLU-77         [-1, 64, 256, 256]               0
        ConvBlock-78         [-1, 64, 256, 256]               0
     DecoderBlock-79         [-1, 64, 256, 256]               0
           Conv2d-80          [-1, 1, 256, 256]              65
          Sigmoid-81          [-1, 1, 256, 256]               0
================================================================
Total params: 31,043,521
Trainable params: 31,043,521
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 990.00
Params size (MB): 118.42
Estimated Total Size (MB): 1109.17
----------------------------------------------------------------
```



圖中藍色的部分進行的操作相同，可以為其定義一個類別 ConvBlock。

https://www.youtube.com/watch?v=T6h-mVVpafI

UNet 是一種經典的語義分割架構，其特點是 **對稱的 Encoder-Decoder 結構** 和 **跳躍連接(Skip Connection)** ，由左側的 *contracting path* 和 右側的 *expansive path* 組成。

- Encoder：
  - 包含 4 個 EncoderBlock，每個 block 由兩個 3×3 卷積層、BatchNorm 和 ReLU函數組成。
  - 每個 EncoderBlock 後跟一個 MaxPool2d 層，逐步減小特徵圖的空間維度。
- Bridge：
  - 位於編碼器和解碼器之間，由一個 ConvBlock 組成。
- Decoder：
  - 包含 4 個 DecoderBlock，每個 block 使用 *轉置卷積(TransposeConv2d)* 進行 *上採樣(Upsampling)* 。
  - 通過 *跳躍連接(Skip Connection)* 將 Encoder 的對應層特徵與上採樣結果連接，然後通過兩個 3×3 卷積層處理。
  - 最大池化層，逐步減小特徵圖的空間維度。
- Final Conv：
  - 使用 1×1 卷積將特徵圖轉換為所需的類別數，在二元分類問題中為 1 個類別。
- Activation：
  - 對於二元分類問題，使用 Sigmoid 激活函數，使得輸出在 $[0, 1]$ 之間。
  - 對於多重分類問題，使用 Softmax 激活函數。

```
class ConvBlock(nn.Module):
    def __init__(self, in_channels=3, out_channels=1):
```

```
        super(ConvBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding='same'),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding='same'),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.conv(x)

class EncoderBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(EncoderBlock, self).__init__()
        self.conv_block = ConvBlock(in_channels, out_channels)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv_block(x)
        p = self.pool(x)
        return x, p

class DecoderBlock(nn.Module):
    def __init__(self, in_channels, skip_channels, out_channels):
        super(DecoderBlock, self).__init__()
        self.upconv = nn.ConvTranspose2d(in_channels, out_channels,
kernel_size=2, stride=2, padding=0)
        self.conv = ConvBlock(in_channels=out_channels + skip_channels,
out_channels=out_channels)

    def forward(self, x, skip_features):
        x = self.upconv(x)
        x = torch.cat((x, skip_features), dim=1)
        x = self.conv(x)
        return x

class UNet(nn.Module):
    def __init__(self, input_channels, n_classes):
        super(UNet, self).__init__()

        self.enc1 = EncoderBlock(input_channels, 64)
        self.enc2 = EncoderBlock(64, 128)
        self.enc3 = EncoderBlock(128, 256)
        self.enc4 = EncoderBlock(256, 512)

        self.bridge = ConvBlock(512, 1024) # bridge

        self.dec1 = DecoderBlock(1024, 512, 512)
        self.dec2 = DecoderBlock(512, 256, 256)
        self.dec3 = DecoderBlock(256, 128, 128)
        self.dec4 = DecoderBlock(128, 64, 64)

        self.final_conv = nn.Conv2d(64, n_classes, kernel_size=1, padding=0)
        self.sigmoid = nn.Sigmoid() # for binary classification
```

```
    def forward(self, x):
        # Encoder
        s1, p1 = self.enc1(x)
        s2, p2 = self.enc2(p1)
        s3, p3 = self.enc3(p2)
        s4, p4 = self.enc4(p3)

        # Bridge
        b1 = self.bridge(p4)

        # Decoder
        d1 = self.dec1(b1, s4)
        d2 = self.dec2(d1, s3)
        d3 = self.dec3(d2, s2)
        d4 = self.dec4(d3, s1)

        # Final Conv
        outputs = self.final_conv(d4)

        # Activation
        outputs = self.sigmoid(outputs)

        return outputs
```

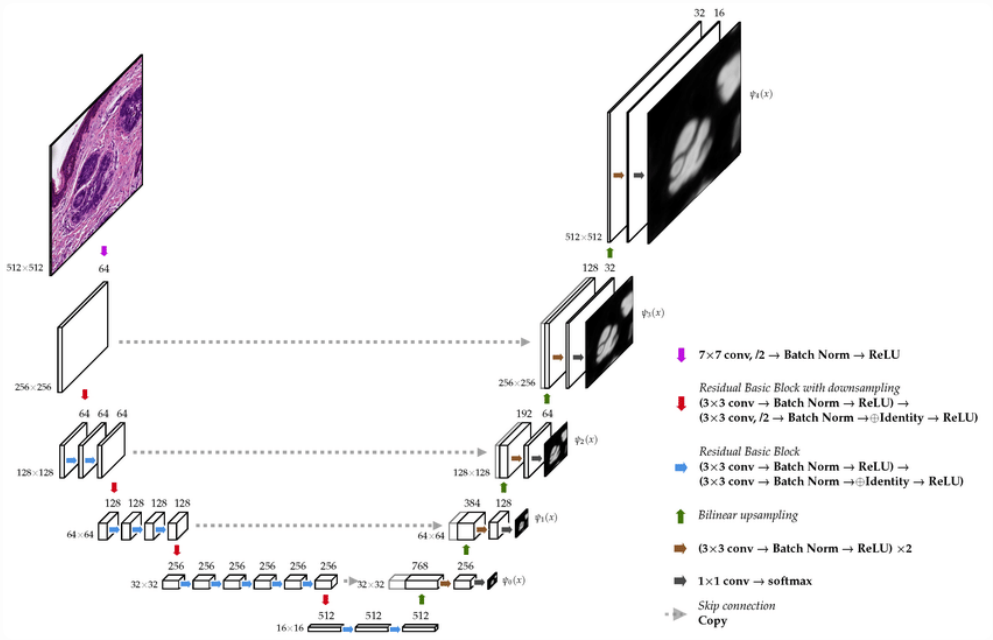
ResNet34_UNet

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 128, 128]           9,408
       BatchNorm2d-2         [-1, 64, 128, 128]             128
              ReLU-3         [-1, 64, 128, 128]               0
         MaxPool2d-4           [-1, 64, 64, 64]               0
            Conv2d-5           [-1, 64, 64, 64]          36,864
       BatchNorm2d-6           [-1, 64, 64, 64]             128
              ReLU-7           [-1, 64, 64, 64]               0
            Conv2d-8           [-1, 64, 64, 64]          36,864
       BatchNorm2d-9           [-1, 64, 64, 64]             128
   ResidualBlock-10           [-1, 64, 64, 64]               0
           Conv2d-11           [-1, 64, 64, 64]          36,864
      BatchNorm2d-12           [-1, 64, 64, 64]             128
             ReLU-13           [-1, 64, 64, 64]               0
           Conv2d-14           [-1, 64, 64, 64]          36,864
      BatchNorm2d-15           [-1, 64, 64, 64]             128
   ResidualBlock-16           [-1, 64, 64, 64]               0
           Conv2d-17           [-1, 64, 64, 64]          36,864
      BatchNorm2d-18           [-1, 64, 64, 64]             128
             ReLU-19           [-1, 64, 64, 64]               0
           Conv2d-20           [-1, 64, 64, 64]          36,864
      BatchNorm2d-21           [-1, 64, 64, 64]             128
   ResidualBlock-22           [-1, 64, 64, 64]               0
    EncoderBlock-23           [-1, 64, 64, 64]               0
           Conv2d-24          [-1, 128, 32, 32]          73,728
      BatchNorm2d-25          [-1, 128, 32, 32]             256
             ReLU-26          [-1, 128, 32, 32]               0
           Conv2d-27          [-1, 128, 32, 32]         147,456
```

```
      BatchNorm2d-28          [-1, 128, 32, 32]             256
            Conv2d-29          [-1, 128, 32, 32]           8,192
      BatchNorm2d-30          [-1, 128, 32, 32]             256
    ResidualBlock-31          [-1, 128, 32, 32]               0
            Conv2d-32          [-1, 128, 32, 32]         147,456
      BatchNorm2d-33          [-1, 128, 32, 32]             256
              ReLU-34          [-1, 128, 32, 32]               0
            Conv2d-35          [-1, 128, 32, 32]         147,456
      BatchNorm2d-36          [-1, 128, 32, 32]             256
    ResidualBlock-37          [-1, 128, 32, 32]               0
            Conv2d-38          [-1, 128, 32, 32]         147,456
      BatchNorm2d-39          [-1, 128, 32, 32]             256
              ReLU-40          [-1, 128, 32, 32]               0
            Conv2d-41          [-1, 128, 32, 32]         147,456
      BatchNorm2d-42          [-1, 128, 32, 32]             256
    ResidualBlock-43          [-1, 128, 32, 32]               0
            Conv2d-44          [-1, 128, 32, 32]         147,456
      BatchNorm2d-45          [-1, 128, 32, 32]             256
              ReLU-46          [-1, 128, 32, 32]               0
            Conv2d-47          [-1, 128, 32, 32]         147,456
      BatchNorm2d-48          [-1, 128, 32, 32]             256
    ResidualBlock-49          [-1, 128, 32, 32]               0
     EncoderBlock-50          [-1, 128, 32, 32]               0
            Conv2d-51          [-1, 256, 16, 16]         294,912
      BatchNorm2d-52          [-1, 256, 16, 16]             512
              ReLU-53          [-1, 256, 16, 16]               0
            Conv2d-54          [-1, 256, 16, 16]         589,824
      BatchNorm2d-55          [-1, 256, 16, 16]             512
            Conv2d-56          [-1, 256, 16, 16]          32,768
      BatchNorm2d-57          [-1, 256, 16, 16]             512
    ResidualBlock-58          [-1, 256, 16, 16]               0
            Conv2d-59          [-1, 256, 16, 16]         589,824
      BatchNorm2d-60          [-1, 256, 16, 16]             512
              ReLU-61          [-1, 256, 16, 16]               0
            Conv2d-62          [-1, 256, 16, 16]         589,824
      BatchNorm2d-63          [-1, 256, 16, 16]             512
    ResidualBlock-64          [-1, 256, 16, 16]               0
            Conv2d-65          [-1, 256, 16, 16]         589,824
      BatchNorm2d-66          [-1, 256, 16, 16]             512
              ReLU-67          [-1, 256, 16, 16]               0
            Conv2d-68          [-1, 256, 16, 16]         589,824
      BatchNorm2d-69          [-1, 256, 16, 16]             512
    ResidualBlock-70          [-1, 256, 16, 16]               0
            Conv2d-71          [-1, 256, 16, 16]         589,824
      BatchNorm2d-72          [-1, 256, 16, 16]             512
              ReLU-73          [-1, 256, 16, 16]               0
            Conv2d-74          [-1, 256, 16, 16]         589,824
      BatchNorm2d-75          [-1, 256, 16, 16]             512
    ResidualBlock-76          [-1, 256, 16, 16]               0
            Conv2d-77          [-1, 256, 16, 16]         589,824
      BatchNorm2d-78          [-1, 256, 16, 16]             512
              ReLU-79          [-1, 256, 16, 16]               0
            Conv2d-80          [-1, 256, 16, 16]         589,824
      BatchNorm2d-81          [-1, 256, 16, 16]             512
    ResidualBlock-82          [-1, 256, 16, 16]               0
            Conv2d-83          [-1, 256, 16, 16]         589,824
      BatchNorm2d-84          [-1, 256, 16, 16]             512
              ReLU-85          [-1, 256, 16, 16]               0
```

```
          Conv2d-86          [-1, 256, 16, 16]         589,824
     BatchNorm2d-87          [-1, 256, 16, 16]             512
    ResidualBlock-88          [-1, 256, 16, 16]               0
     EncoderBlock-89          [-1, 256, 16, 16]               0
          Conv2d-90            [-1, 512, 8, 8]       1,179,648
     BatchNorm2d-91            [-1, 512, 8, 8]           1,024
            ReLU-92            [-1, 512, 8, 8]               0
          Conv2d-93            [-1, 512, 8, 8]       2,359,296
     BatchNorm2d-94            [-1, 512, 8, 8]           1,024
          Conv2d-95            [-1, 512, 8, 8]         131,072
     BatchNorm2d-96            [-1, 512, 8, 8]           1,024
    ResidualBlock-97            [-1, 512, 8, 8]               0
          Conv2d-98            [-1, 512, 8, 8]       2,359,296
     BatchNorm2d-99            [-1, 512, 8, 8]           1,024
           ReLU-100            [-1, 512, 8, 8]               0
         Conv2d-101            [-1, 512, 8, 8]       2,359,296
    BatchNorm2d-102            [-1, 512, 8, 8]           1,024
   ResidualBlock-103            [-1, 512, 8, 8]               0
         Conv2d-104            [-1, 512, 8, 8]       2,359,296
    BatchNorm2d-105            [-1, 512, 8, 8]           1,024
           ReLU-106            [-1, 512, 8, 8]               0
         Conv2d-107            [-1, 512, 8, 8]       2,359,296
    BatchNorm2d-108            [-1, 512, 8, 8]           1,024
   ResidualBlock-109            [-1, 512, 8, 8]               0
    EncoderBlock-110            [-1, 512, 8, 8]               0
         Conv2d-111           [-1, 1024, 8, 8]       4,718,592
    BatchNorm2d-112           [-1, 1024, 8, 8]           2,048
           ReLU-113           [-1, 1024, 8, 8]               0
      MaxPool2d-114           [-1, 1024, 4, 4]               0
ConvTranspose2d-115            [-1, 512, 8, 8]       2,097,664
         Conv2d-116            [-1, 512, 8, 8]       4,718,592
    BatchNorm2d-117            [-1, 512, 8, 8]           1,024
           ReLU-118            [-1, 512, 8, 8]               0
         Conv2d-119            [-1, 512, 8, 8]       2,359,296
    BatchNorm2d-120            [-1, 512, 8, 8]           1,024
           ReLU-121            [-1, 512, 8, 8]               0
   DecoderBlock-122            [-1, 512, 8, 8]               0
ConvTranspose2d-123          [-1, 256, 16, 16]         524,544
         Conv2d-124          [-1, 256, 16, 16]       1,179,648
    BatchNorm2d-125          [-1, 256, 16, 16]             512
           ReLU-126          [-1, 256, 16, 16]               0
         Conv2d-127          [-1, 256, 16, 16]         589,824
    BatchNorm2d-128          [-1, 256, 16, 16]             512
           ReLU-129          [-1, 256, 16, 16]               0
   DecoderBlock-130          [-1, 256, 16, 16]               0
ConvTranspose2d-131          [-1, 128, 32, 32]         131,200
         Conv2d-132          [-1, 128, 32, 32]         294,912
    BatchNorm2d-133          [-1, 128, 32, 32]             256
           ReLU-134          [-1, 128, 32, 32]               0
         Conv2d-135          [-1, 128, 32, 32]         147,456
    BatchNorm2d-136          [-1, 128, 32, 32]             256
           ReLU-137          [-1, 128, 32, 32]               0
   DecoderBlock-138          [-1, 128, 32, 32]               0
ConvTranspose2d-139           [-1, 64, 64, 64]          32,832
         Conv2d-140           [-1, 64, 64, 64]          73,728
    BatchNorm2d-141           [-1, 64, 64, 64]             128
           ReLU-142           [-1, 64, 64, 64]               0
         Conv2d-143           [-1, 64, 64, 64]          36,864
```

```
      BatchNorm2d-144           [-1, 64, 64, 64]             128
             ReLU-145           [-1, 64, 64, 64]               0
    DecoderBlock-146           [-1, 64, 64, 64]               0
 ConvTranspose2d-147         [-1, 64, 128, 128]          16,448
          Conv2d-148         [-1, 64, 128, 128]          73,728
     BatchNorm2d-149         [-1, 64, 128, 128]             128
             ReLU-150         [-1, 64, 128, 128]               0
          Conv2d-151         [-1, 64, 128, 128]          36,864
     BatchNorm2d-152         [-1, 64, 128, 128]             128
             ReLU-153         [-1, 64, 128, 128]               0
    DecoderBlock-154         [-1, 64, 128, 128]               0
 ConvTranspose2d-155         [-1, 64, 256, 256]          16,448
          Conv2d-156          [-1, 1, 256, 256]             576
================================================================
Total params: 38,340,032
Trainable params: 38,340,032
Non-trainable params: 0
Input size (MB): 0.75
Forward/backward pass size (MB): 243.88
Params size (MB): 146.26
Estimated Total Size (MB): 390.88
----------------------------------------------------------------
```



註：圖中的長寬為本次實驗的兩倍

Deeply Supervised UNet for Semantic Segmentation to Assist Dermatopathological Assessment of Basal Cell Carcinoma

ResNet34_UNet 是一種結合了 ResNet34 作為 Encoder 和 UNet 風格 Decoder 的混合架構。

- Encoder（基於 ResNet34）：
  - 初始層：7×7 卷積，BatchNorm、ReLU、MaxPool2d。
  - 4 個 EncoderBlock，每個 block 包含多個 ResidualBlock，事實上，這就是 ResNet34 的 Layer。

- Bridge：
  - 位於編碼器和解碼器之間，由一個 ConvBlock 組成，包含了 3×3 卷積層、BatchNorm、ReLU、MaxPool2d。
- Decoder：
  - 5 個 DecoderBlock，每個 block 使用 *轉置卷積(TransposeConv2d)* 進行 *上採樣(Upsampling)* 。
  - 將上採樣結果與 Encoder 對應層的特徵圖連接。
  - 每個 block 包含兩個 3×3 卷積層，BatchNorm 和 ReLU。
- 最終層：
  - 使用轉置卷積進行最後的上採樣，然後是 1×1 卷積，將特徵圖轉換為所需的類別數，在二元分類問題中為 1 個類別。

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, shortcut=None):
        super(ResidualBlock, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride,
padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )
        self.shortcut = shortcut

    def forward(self, x):
        out = self.block(x)
        residual = x if self.shortcut is None else self.shortcut(x)
        out += residual
        # out = self.relu(out)
        return F.relu(out)

class EncoderBlock(nn.Module):
    def __init__(self, in_channels, out_channels, n_blocks, stride,
is_shortcut=True):
        super(EncoderBlock, self).__init__()

        if is_shortcut:
            shortcut = nn.Sequential(
                        nn.Conv2d(in_channels, out_channels, kernel_size=1,
stride=stride, bias=False),
                        nn.BatchNorm2d(out_channels))
        else:
            shortcut = None

        layers = []
        layers.append(ResidualBlock(in_channels, out_channels, stride,
shortcut))
```

```
        for _ in range(1, n_blocks):
            layers.append(ResidualBlock(out_channels, out_channels))

        self.blocks = nn.Sequential(*layers)

    def forward(self, x):
        out = self.blocks(x)
        return out

class DecoderBlock(nn.Module):
    # def __init__(self, in_channels, out_channels):
    def __init__(self, in_channels, out_channels, up_in_channels=None,
up_out_channels=None):
        super(DecoderBlock, self).__init__()
        if up_in_channels is None:
            up_in_channels = in_channels
        if up_out_channels is None:
            up_out_channels = out_channels
        self.up = nn.ConvTranspose2d(up_in_channels, up_out_channels,
kernel_size=2, stride=2)
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1,
bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    # x1-upconv , x2-downconv
    def forward(self, x1, x2):
        x1 = self.up(x1)
        x = torch.cat([x1, x2], dim=1)
        return self.block(x)

class ResNet34_UNet(nn.Module):
    def __init__(self, in_channels, n_classes):
        super(ResNet34_UNet, self).__init__()
        self.enc1 = nn.Sequential(
            nn.Conv2d(in_channels, 64, kernel_size=7, stride=2, padding=3,
bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
        )
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.enc2 = EncoderBlock(64, 64, 3, 1, is_shortcut=False)
        self.enc3 = EncoderBlock(64, 128, 4, 2)
        self.enc4 = EncoderBlock(128, 256, 6, 2)
        self.enc5 = EncoderBlock(256, 512, 3, 2)

        self.bridge = nn.Sequential(
            nn.Conv2d(512, 1024, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(1024),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
```

```
        )
        self.dec1 = DecoderBlock(1024, 512)
        self.dec2 = DecoderBlock(512, 256)
        self.dec3 = DecoderBlock(256, 128)
        self.dec4 = DecoderBlock(128, 64)
        self.dec5 = DecoderBlock(128, 64, 64, 64)

        self.lastlayer = nn.Sequential(
            nn.ConvTranspose2d(in_channels=64, out_channels=64, kernel_size=2,
stride=2),
            nn.Conv2d(64, n_classes, kernel_size=3, padding=1, bias=False)
        )

    def forward(self, x):
        enc1 = self.enc1(x)
        maxpool = self.maxpool(enc1)
        enc2 = self.enc2(maxpool)
        enc3 = self.enc3(enc2)
        enc4 = self.enc4(enc3)
        enc5 = self.enc5(enc4)

        center = self.bridge(enc5)

        dec1 = self.dec1(center, enc5)
        dec2 = self.dec2(dec1, enc4)
        dec3 = self.dec3(dec2, enc3)
        dec4 = self.dec4(dec3, enc2)
        dec5 = self.dec5(dec4, enc1)

        out = self.lastlayer(dec5)
        return out
```

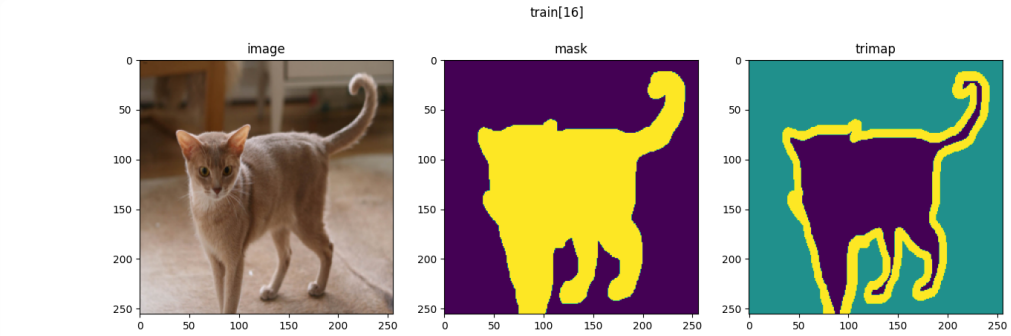
C. Anything more you want to mention

在實作 UNet 的部分，相較於原始論文，有一些不同的地方：

- 使用了當時(2015)還不流行的 BatchNorm
- 在卷積的部分做了 padding ，以確保輸入和輸出的形狀一致，並且在使用 skip connection 時，不用進行裁減，可以保留更多的特徵。

3. Data Preprocessing (20%)

A. How you preprocessed your data?

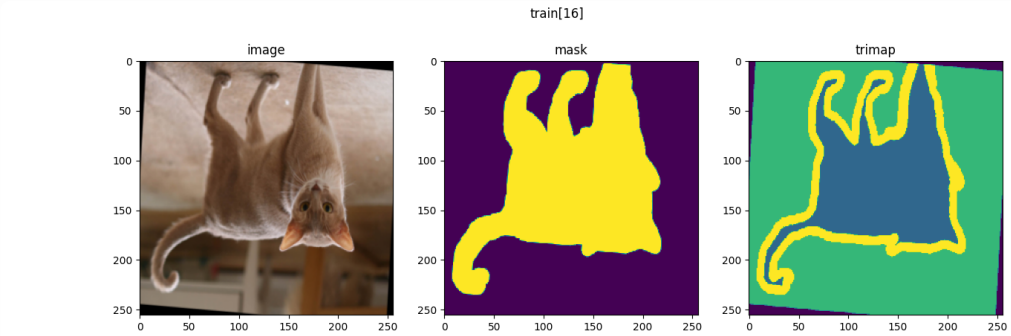


每個 sample 由 image、trimap、mask 組成，其中 image 是原始圖像、trimap 是三元圖，mask 是二元圖，以 mask 作為 ground truth。

在本次 Lab 中，我們對 `Oxford-IIIT Pet Dataset` 進行了以下預處理步驟，除了最後一項外，都已經在助教提供的程式碼中完成：

- 使用 dataset 中的 `trainval.txt` 和 `test.txt` 來分別切割 train set, validation set 和 test set。
  - train set 和 validation set 的切割比例為 9:1，將索引為 10 的倍數的資料分到 validset 中，其餘分到 trainset 中。
  - test set 則是直接使用 `test.txt` 中的所有資料。
- 使用 `_preprocess_mask` 函數將 trimap 轉換為 mask 的二值圖像。
- 將所有圖片 `resize` 為 256×256 的大小。
- 將 HWC (height, width, channel) 轉換為 CHW (channel, height, width) 的格式。
- 由於作為 ground truth 的 mask 是 0 或 1 的二值圖像，因此也將輸入的 image 從 $[0, 255]$ 轉換為 $[0, 1]$ 的形式。

B. What makes your method unique?



經過 transform 函數之後的 train[16]，包含了上下翻轉、左右翻轉、旋轉一定角度。

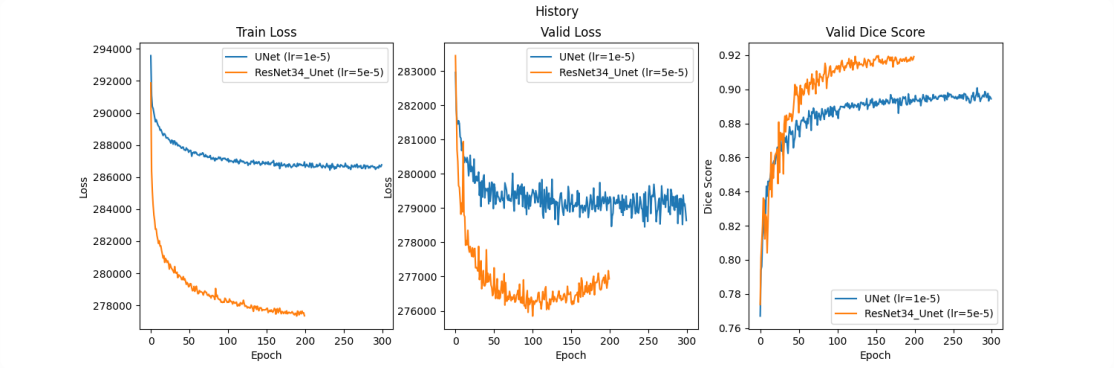
在訓練時使用了一些資料增強(Data Augmentation)的方法：

- 使用 `opencv` 將圖片進行水平或垂直翻轉。
- 使用 `opencv` 將圖片旋轉 $[-15, 15]$ 度。

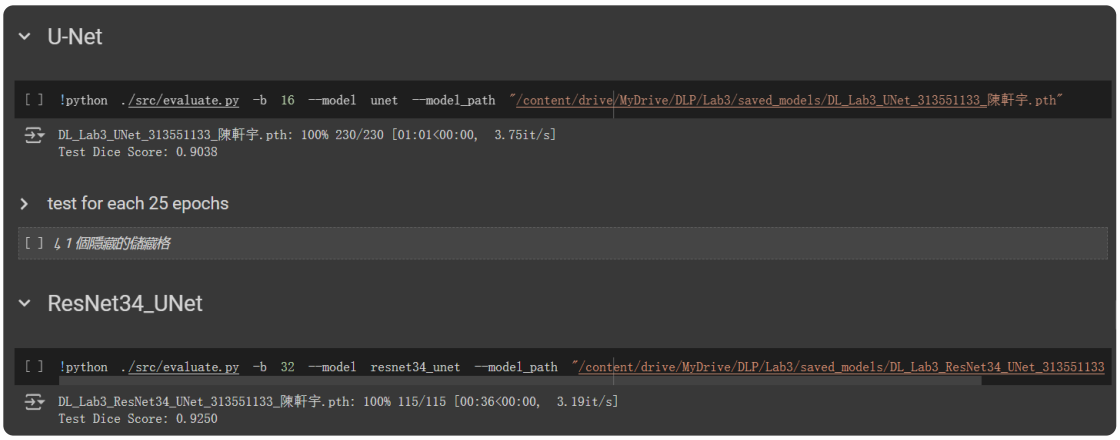
```python
def rotate(image, angle, center=None, scale=1.0):
    (h, w) = image.shape[:2] # grab the dimensions of the image

    if center is None:
        center = (w // 2, h // 2)

    M = cv2.getRotationMatrix2D(center, angle, scale)
    rotated = cv2.warpAffine(image, M, (w, h))

    return rotated

def transform(sample):
    deg = np.random.randint(-15, 16) # [-15, 15]
    flip1 = np.random.rand() # [0, 1]
    flip2 = np.random.rand() # [0, 1]

    for key in sample.keys():
        sample[key] = rotate(sample[key], deg)
        if flip1 > 0.5:  # flipping around the x-axis (vertically)
            sample[key] = cv2.flip(sample[key], 0)
        if flip2 > 0.5: # flipping around the y-axis (horizontally)
            sample[key] = cv2.flip(sample[key], 1)
    return sample
```

C. Anything more you want to mention

4. Analyze on the experiment results (20%)



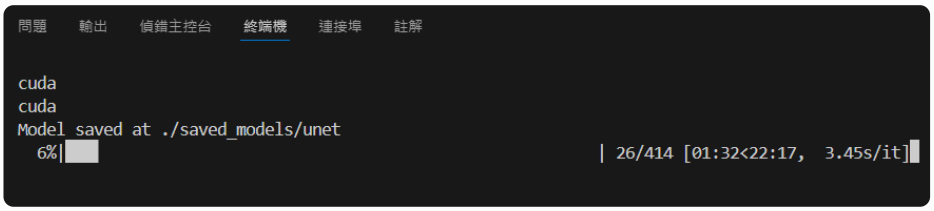
實驗結果，在 UNet 與 ResNet34_UNet 上分別訓練了 300 和 200 個 epochs。



在 test set 上的 dice score 分別為 0.9038 和 0.9250。

A. What did you explore during the training process?

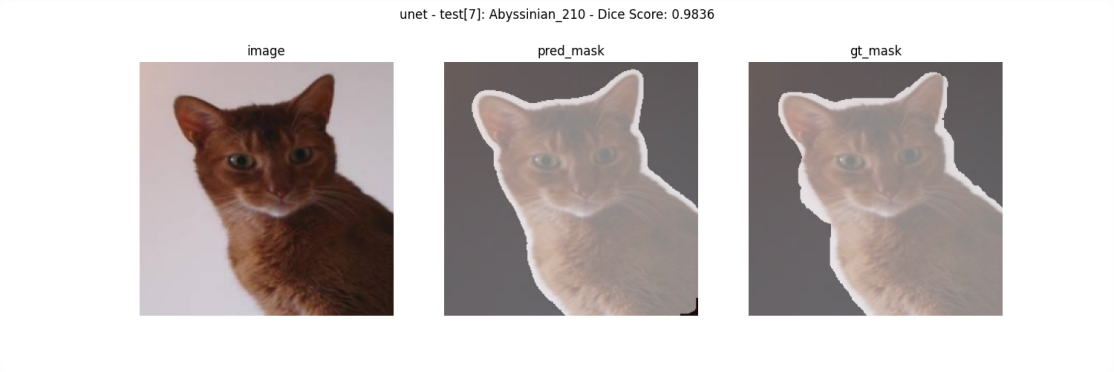
從實驗結果中注意到不管是 UNet 還是 ResNet34_UNet，都很容易發生 overfitting 的現象，雖然在 train set 上的 loss 持續下降，但在 valid set 上的 loss 卻出現了不升反降的情況。但即使如此，最終 UNet 和 ResNet34_UNet 在 test set 上的 dice score 仍然可以達到 0.9 以上 。



It takes so much time to train UNet and ResNet34_UNet on my local GPU.

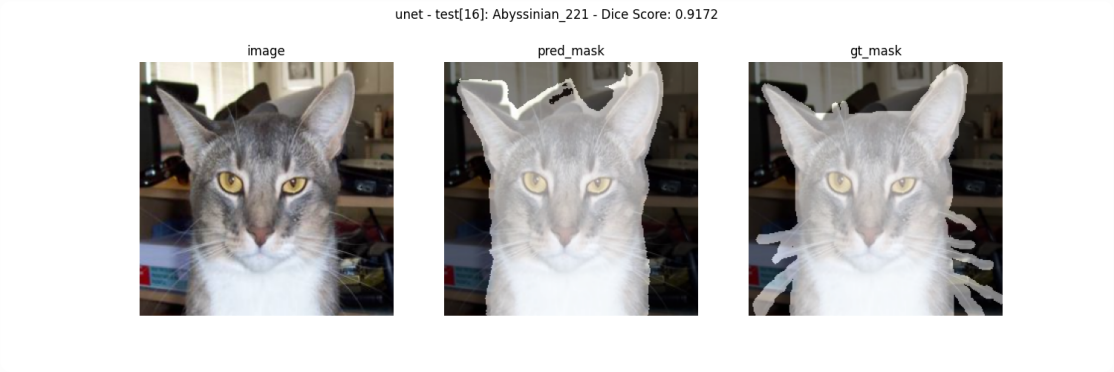
此外，根據 `torchsummary` 的結果，在 UNet 和 ResNet34_UNet 上的總參數量分別為 390.88MB 和 1109.17MB，使得這次訓練時間相對長了很多，若使用本地GPU，可能會 train 到下一個 Lab 的 deadline，最後不得不啟用 Colab Pro 。但即使如此，在訓練 UNet 時，以 Tesla L4 訓練 300 個 epochs 仍需花約 10 個小時的時間。

B. Found any characteristics of the data?



簡單的單色背景，可以比較輕易的區分出寵物和背景。

寵物的部份身體（尾巴）沒有出現在圖片中，且寵物顏色與背景顏色相近且背景為複雜的室內場景。



寵物的毛髮邊緣很難被精準判讀。

在 `Oxford-IIIT Pet Dataset` 中，有以下幾個值得注意的特徵：

1. 品種多樣性：
    - 根據網頁上的描述，此資料集包含了 37 種不同的寵物品種，其中包括 25 種狗和 12 種貓，每種品種都有約 200 張圖片。
    - 在不同的品種間可能存在一些型態差異，影響模型的泛化能力。
2. 姿勢變化：
    - 寵物資料集中有多種姿勢，例如正面、側面、躺臥等。
    - 某些姿勢（如側臥）使得寵物的部分身體特徵難以辨識。
3. 背景複雜度：
    - 圖片背景的變化很大，從簡單的單色背景到複雜的室內外場景都有。
    - 部分圖片中，寵物與背景的顏色相近，增加了分割前後景的難度。
4. 遮擋問題：
    - 在一些圖片中，寵物被部分遮擋（如被家具或其他物體擋住）。
    - 這增加了完整分割寵物輪廓的難度。
5. 毛髮細節:
    - 不同品種的寵物毛髮特徵差異很大，如長毛、短毛、捲毛等。
    - 精確分割毛髮邊緣是一個挑戰，特別是對於長毛品種。
    - 此外，貓咪的鬍鬚若與背景的色彩相近，則可能會影響分割結果。

C. Anything more you want to mention

雖然在訓練 UNet 時並沒有在 validation set 上看到 > 0.9 的 dice score，但在測試集上的 dice score 卻還是勉強達到了 0.9，很幸運能夠省去一些重新訓練的時間。

5. Execution command (0%)

實際訓練是在 Google Colab 上執行，而指令上的區別除了 batch size 外，只在於路徑的不同，在 Google Colab 上訓練和推論時我會直接將檔案存入 Google Drive 中。

A. The command and parameters for the training process

UNet

Using the following command to train on Google Colab:

```
!python ./src/train.py --model unet -e 300 -b 32 -lr 1e-5 --save_path
"/content/drive/MyDrive/DLP/Lab3/saved_models"
```

Using the following command to train on Local Machine:

```
python ./src/train.py --model unet -e 300 -b 16 -lr 1e-5 --save_path
"./saved_models/"
```

ResNet34_UNet

Using the following command to train on Google Colab:

```
!python ./src/train.py --model resnet34_unet -e 200 -lr 5e-5 -b 32 --save_path
"/content/drive/MyDrive/DLP/Lab3/saved_models"
```

Using the following command to train on Local Machine:

```
python ./src/train.py --model resnet34_unet -e 200 -lr 5e-5 -b 16 --save_path
"./saved_models/"
```

B. The command and parameters for the inference process

正如前述所述，在推論時可以選擇需要推論單張圖片或是所有圖片，若只推論單張圖片，則可以指定要推論的圖片的索引，其結果會如 4.B 中展示的結果圖片。

```
usage: inference.py [-h] [--data_path DATA_PATH] [--batch_size BATCH_SIZE] [--
model MODEL]
                    [--model_path MODEL_PATH] [--output_path OUTPUT_PATH] [--
idx IDX]

Predict masks from input images

options:
  -h, --help            show this help message and exit
  --data_path DATA_PATH
                        path of the input data
  --batch_size BATCH_SIZE, -b BATCH_SIZE
                        batch size
  --model MODEL         model to use (unet/resnet34_unet)
  --model_path MODEL_PATH
```

```
                        path to load the model
  --output_path OUTPUT_PATH
                        path to save the output
  --idx IDX, -i IDX     index of the image to predict
```

UNet

Google Colab

Inference single image and show the result on **Google Colab**:

```
!python ./src/inference.py -i 116 --model unet --model_path
"/content/drive/MyDrive/DLP/Lab3/saved_models/DL_Lab3_UNet_313551133_陳軒宇.pth"
--output_path "/content/drive/MyDrive/DLP/Lab3/output"
display(Image('/content/drive/MyDrive/DLP/Lab3/output/unet_116.png'))
```

Inference all images and save the result to output_path on **Google Colab**:

```
!python ./src/inference.py -b 32 --model unet --model_path
"/content/drive/MyDrive/DLP/Lab3/saved_models/DL_Lab3_UNet_313551133_陳軒宇.pth"
--output_path "/content/drive/MyDrive/DLP/Lab3/output"
```

Local Machine

Inference single image and save the result to output_path on **local machine**:

```
python ./src/inference.py -i 116 --model unet --model_path
"./saved_models/DL_Lab3_UNet_313551133_陳軒宇.pth" --output_path "./output"
```

Inference all images and save the result to output_path on **local machine**:

```
python ./src/inference.py -b 8 --model unet --model_path
"./saved_models/DL_Lab3_UNet_313551133_陳軒宇.pth" --output_path "./output"
```

ResNet34_UNet

Google Colab

Inference single image and show the result on **Google Colab**:

```
!python ./src/inference.py -i 116 --model resnet34_unet --model_path
"/content/drive/MyDrive/DLP/Lab3/saved_models/DL_Lab3_ResNet34_UNet_313551133_陳
軒宇.pth" --output_path "/content/drive/MyDrive/DLP/Lab3/output"
display(Image('/content/drive/MyDrive/DLP/Lab3/output/resnet34_unet_116.png'))
```

Inference all images and save the result to Google Drive on **Google Colab**:

```
!python ./src/inference.py -b 32 --model resnet34_unet --model_path
"/content/drive/MyDrive/DLP/Lab3/saved_models/DL_Lab3_ResNet34_UNet_313551133_陳
軒宇.pth" --output_path "/content/drive/MyDrive/DLP/Lab3/output"
```

Local Machine

Inference single image and save the result to output_path on **local machine**:

```
python ./src/inference.py -i 116 --model resnet34_unet --model_path
"./saved_models/DL_Lab3_ResNet34_UNet_313551133_陳軒宇.pth" --output_path
"./output"
```

Inference all images and save the result to output_path on **local machine**:

```
python ./src/inference.py -b 16 --model resnet34_unet --model_path
"./saved_models/DL_Lab3_ResNet34_UNet_313551133_陳軒宇.pth" --output_path
"./output"
```

6. Discussion (20%)

A. What architecture may bring better results?

雖然實驗結果是 ResNet34_UNet 的 dice score 相對較高，但我在 UNet 和 ResNet34_UNet 的學習率分別為 1×10^{-5} 和 5×10^{-5}。雖然沒有展示，但我也使用了學習率為 1×10^{-5} 去訓練 ResNet34_UNet ，發現最後的結果不太好。

在沒有做完整對照實驗($$$$$)的情況下，我也不能判定 UNet 在把學習率提升為 5×10^{-5} 時的結果是否會比 ResNet34_UNet 的結果好。

只從結果來看，ResNet34_UNet收斂的速度較快，且最後的 dice score 達到了 0.9250 ，相對於 UNet 的 0.9038，表現得比較好。

B. What are the potential research topics in this task?

1. 多元分割：
    - 將二元分割應用到更多的類別，可以進一步將寵物分割為更細緻的部位，例如寵物的頭部、身體、尾巴等。
2. 3D UNet：
    - 增加輸入的維度，將分類問題應用到三維空間上。
3. 極端條件下的分割：
    - 研究如何提高模型在極端光照、部分遮擋、運動模糊等條件下的分割性能。
4. 實時分割：

- 從實驗中可以發現，在做訓練和推論時對於裝置的GPU性能有一定要求。
- 因此可以開發輕量但高效的分割模型，使其能夠在行動裝置或實況流中執行。
- 研究模型壓縮和加速技術在保持分割正確率的同時提高運行速度。

C. Anything more you want to mention