

1. Overview of your lab 3 (10%)

本次lab為使用U-Net、ResNet34+U-Net兩種模型針對貓狗資料集進行二值化的語義分割，此外因為需考量設備限制故將img size設成256*256，batch size = 4，完成本次Lab。由於資料預處理部分，雖有範例程式將貓或狗設為同一類(1)，背景設為另一類(0)，但考量增加影像多元性，故隨機將照片進行鏡射，並且將照片調整亮度以提高昏暗照片時輪廓較不明顯的問題。透過本方法，最終U-Net、ResNet34+U-Net兩個模型分別在測試時Dice Score達到0.9260、0.9257。

參考資料：

- [1] https://blog.csdn.net/jh_chen/article/details/138261503
- [2] https://blog.csdn.net/weixin_43229348/article/details/122720011
- [3] <https://arxiv.org/abs/1512.03385>
- [4]
https://www.researchgate.net/publication/359463249_Deep_learning-based_pelvic_levator_hiatus_segmentation_from_ultrasound_images
- [5] <https://arxiv.org/abs/1505.04597v1>

2. Implementation Details (30%)

A. Details of your training, evaluating, inferencing code

(1) training code:

```
import os
import torch
import argparse
from torch.utils.data import DataLoader
from torch.optim.lr_scheduler import ReduceLROnPlateau
from models.resnet34_unet import ResNet34_Unet
from models.unet import UNet
from oxford_pet import load_dataset
from evaluate import evaluate
from utils import dice_score, dice_loss, plot_training_history
import time # Import time module

if not os.path.exists('saved_models/'):
    os.mkdir('saved_models')

def train(args, train_dataset, valid_dataset):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = ResNet34_Unet().to(device) if 'resnet34' in args.model else UNet().to(device)

    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr, weight_decay=5e-4)
    scheduler = ReduceLROnPlateau(optimizer, mode='max', factor=0.1, patience=5)
    criterion = dice_loss

    train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True)
    valid_loader = DataLoader(valid_dataset, batch_size=args.batch_size, shuffle=False)
    print(f'Training {args.model} with epochs={args.epochs}, batch size={args.batch_size},')

    best_dice = 0.0 # Initialize the best dice score
    start_time = time.time() # Record the starting time

    train_loss_list = []
    valid_loss_list = []
    train_dice_list = []
    valid_dice_list = []

    best_valid_loss = float('inf')
    epochs_no_improve = 0 # Number of epochs with no improvement
    early_stop_patience = 10 # Early stopping patience
    , initial learning rate={args.lr:.4f}'')
```

前者導入相關套件，並且檢查是否具有路徑用來儲存權重檔案。

後者設定優化器為Adam並且設定weight_decay為0.0005，並且設定當超過5個epoch沒有降低loss時，lr會*factor，此外也設定early stopping(當超過10個epoch未下降時，則中斷訓練避免overfitting)，損失函數因為評估標準為dice故使用dice_loss。接著初始化train、valid的loss、dice的list，用於訓練結束後印出損失函數圖。

```

for epoch in range(args.epochs):
    model.train()
    epoch_loss = 0
    epoch_dice = 0
    num_batches = len(train_loader)
    epoch_start_time = time.time() # Start time of the current epoch

    for i, (data, mask) in enumerate(train_loader):
        inputs = data.to(device)
        masks = mask.to(device)
        optimizer.zero_grad()#clean gradient
        outputs = model(inputs)#predict_mask
        loss = criterion(outputs, masks)
        loss.backward()
        optimizer.step()#update

        batch_dice = dice_score(outputs, masks).item()#calculate current
        epoch_loss += loss.item()
        epoch_dice += batch_dice
        current_lr = optimizer.param_groups[0]['lr']
        print(f'Epoch {epoch+1}/{args.epochs}, Batch {i+1}/{num_batches}

```

- Loss: {loss.item():.4f}, Dice: {batch_dice:.4f}, LR: {current_lr:.1E}')

```

train_loss = epoch_loss / num_batches
train_dice = epoch_dice / num_batches
valid_dice, valid_loss = evaluate(model, valid_loader, device)
scheduler.step(valid_dice)

train_loss_list.append(train_loss)
valid_loss_list.append(valid_loss)
train_dice_list.append(train_dice)
valid_dice_list.append(valid_dice)

print(f'Epoch {epoch+1} Summary - Training Loss: {train_loss:.4f},

```

, Training Dice: {train_dice:.4f}, Validation Loss: {valid_loss:.4f}, Validation Dice: {valid_dice:.4f}')

訓練Epoch迴圈，計算並印出每個batch的Dice值、Dice_loss並且印出當前lr。此外，把每個batch的Dice、Dice_loss加總並且/總batch數量，並存放於list中以便製圖。

```

if valid_dice > best_dice:
    best_dice = valid_dice
    torch.save(model.state_dict(), 'saved_models/best.pth')
    print(f'Saving best model with Dice {best_dice:.4f}')

if valid_loss < best_valid_loss:
    best_valid_loss = valid_loss
    epochs_no_improve = 0
else:
    epochs_no_improve += 1

epoch_duration = time.time() - epoch_start_time
print(f'Epoch {epoch+1} Duration: {epoch_duration:.2f} seconds')

if epochs_no_improve >= early_stop_patience:
    print(f'Early stopping triggered. No improvement in validation loss for {early_stop_patience} epochs.')
    break

return train_loss_list, valid_loss_list, train_dice_list, valid_dice_list

```

若是當前valid_dice筆最大的valid_dice大時，更新並儲存，此外並記錄當前valid_loss是否低於最低值，若是則+1，當 ≥ 10 時，則觸發Early Stopping。

```

def get_args():
    parser = argparse.ArgumentParser(description='Train the UNet on images and target masks')
    parser.add_argument("-g", "--gpu", help="gpu id", default=0, type=int)
    parser.add_argument("-n", "--name", help="Experiment name", default="", type=str)
    parser.add_argument('--model', '-m', type=str, default='unet', help='target model')
    parser.add_argument('--data_path', '-path', type=str, default=r'C:\Users\蘇柏叡\Desktop\DLProject\dataset', help='path of dataset')
    parser.add_argument('--epochs', '-e', type=int, default=50, help='number of epochs')
    parser.add_argument('--batch_size', '-b', type=int, default=4, help='batch size')
    parser.add_argument('--learning-rate', '-lr', type=float, default=1e-4, dest='lr', help='learning rate')
    return parser.parse_args()

if __name__ == "__main__":
    args = get_args()
    train_dataset = load_dataset(args.data_path, 'train')
    valid_dataset = load_dataset(args.data_path, 'valid')
    train_loss, valid_loss, train_dice, valid_dice = train(args, train_dataset, valid_dataset)
    plot_training_history(train_loss, valid_loss, train_dice, valid_dice)

```

args function和原始碼大同小異，僅調整資料路徑以及新增繪製訓練過程圖所需的程式。

(2)evaluating code

```
import torch
from utils import dice_score, dice_loss

def evaluate(model, valid_loader, device):
    model.eval()
    total_dice = 0.0
    total_loss = 0.0
    criterion = dice_loss

    with torch.no_grad():
        for inputs, targets in valid_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)#predict_mask
            loss = criterion(outputs, targets)
            dice = dice_score(outputs, targets)

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

    num_samples = len(valid_loader)
    average_dice = total_dice / num_samples
    average_loss = total_loss / num_samples

    return average_dice, average_loss
```

引入計算dice、dice_loss的函式，並開始計算輸出的mask和真實值的loss並進行評估，此外計算總共的loss、dice後除以驗證集長度即為驗證集的平均dice value、dice loss。

(3)inference.py

```
import argparse
import torch
from utils import dice_score
from oxford_pet import load_dataset
from models.resnet34_unet import ResNet34_Unet
from models.unet import UNet
from torch.utils.data import DataLoader

def get_args():
    parser = argparse.ArgumentParser(description='Predict masks from input images')
    parser.add_argument('--model', default='resnet34', help='model type: resnet34 or unet')
    parser.add_argument('--weights', default='saved_models/best.pth', help='path to stored model weights')
    parser.add_argument('--data_path', type=str, required=True, help='path to the input data')
    parser.add_argument('--batch_size', '-b', type=int, default=8, help='batch size')
    parser.add_argument("--g", "--gpu", help="gpu id", default=0, type=int)
    return parser.parse_args()
```

引入相關套件，get_args和原碼是大同小異！

```

def test(model, test_dataset, batch_size, device):
    model.eval()
    test_loader = DataLoader(test_dataset, batch_size = batch_size)
    test_dice = 0

    #criterion = dice_loss測試集不需要loss所以不用
    num_samples = len(test_loader)
    with torch.no_grad():
        for inputs, targets in test_loader: #inputs: image, targets: mask
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs) #predict_mask
            test_dice += dice_score(outputs, targets)
    test_dice /= num_samples
    return test_dice

```

因為在testing時，不用計算其損失值，所以僅需要知道測試集長度並且拿evaluate.py稍作修改，即計算每個batch的dice_score並且加總取平均即可。

```

if __name__ == '__main__':
    args = get_args()
    test_dataset = load_dataset(args.data_path, 'test')
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    if args.model == 'resnet34_unet':
        model = ResNet34_Unet().to(device)
    elif args.model == 'unet':
        model = UNet().to(device)
    else:
        raise ValueError("Unknown model type. Please choose 'resnet34_unet' or 'unet'.")

    model.load_state_dict(torch.load(args.weights))
    model.to(device)

    test_dice = test(model, test_dataset, args.batch_size, device)
    print(f'{args.model} Testing Dice Score:{test_dice:.4f}')

```

主程式主要用來導入測試集，並導入權重、呈現測試結果值。

A. Details of your model (UNet & ResNet34_UNet)

```
import torch
import torch.nn as nn

class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.model = 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)
        )

    def forward(self, x):
        return self.model(x)
```

針對上採樣、下採樣皆做兩次convolution的特點寫成DoubleConv，內容主要是呈現kernel size為3做卷積，此外為了要保持圖片大小不變，故需要做padding。

```
class DownSampling(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownSampling, self).__init__()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv = DoubleConv(in_channels, out_channels)

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

class UpSampling(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UpSampling, self).__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, x, skip):
        x = self.up(x)
        x = torch.cat([x, skip], dim=1)
        return self.conv(x)
```

下採樣部分(兩次卷積一次MaxPooling)，由於是 2×2 最大池化且步長設2，以達成縮小為原尺寸的 $1/2$ 。另外，上採樣使用ConvTranspose2d做逆卷積並且將特徵圖通道數量 $/2$ ，而在forward處x經過轉置卷積後會把尺寸變兩倍，之後透過torch.cat把上採樣後的結果和來自下採樣路徑對應的feature map進行跳躍連接。

```

class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        self.conv1 = DoubleConv(3, 64)
        self.down1 = DownSampling(64, 128)
        self.down2 = DownSampling(128, 256)
        self.down3 = DownSampling(256, 512)
        self.down4 = DownSampling(512, 1024)
        self.up1 = UpSampling(1024, 512)
        self.up2 = UpSampling(512, 256)
        self.up3 = UpSampling(256, 128)
        self.up4 = UpSampling(128, 64)
        self.final_conv = nn.Conv2d(64, 1, kernel_size=1)

    def forward(self, x):
        c1 = self.conv1(x)
        d1 = self.down1(c1)
        d2 = self.down2(d1)
        d3 = self.down3(d2)
        d4 = self.down4(d3)
        u1 = self.up1(d4, d3)
        u2 = self.up2(u1, d2)
        u3 = self.up3(u2, d1)
        u4 = self.up4(u3, c1)
        out = self.final_conv(u4)
        return torch.sigmoid(out)

```

這邊是U-Net的本體，分別輸入model的channel並且完成各4次上、下採樣，而最後輸出1個channel是因為本task為binary(意即num_class = 1，不含背景)並且在上採樣過程做跳躍連接是U-Net的特性。

ResNet34+U-Net

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class BasicBlock(nn.Module):#ResNet最一開始的block
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

    ##接下來做shortcut的部分
    if stride != 1 or in_channels != out_channels:
        ##shortcut當stride不為1或是in_channels不等於out_channels時，要使用1x1的convolution來調整
        self.shortcut = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(out_channels)
        )
    else:
        self.shortcut = nn.Identity()#不做任何調整

    def forward(self, x):#順序是conv1->bn1->relu->conv2->bn2->shortcut->relu
        residual = self.shortcut(x)
        x = self.relu(self.bn1(self.conv1(x)))#對x做convolution->batch normalization->relu
        x = self.bn2(self.conv2(x))
        x += residual
        x = self.relu(x)
        return x
```

根據架構圖做ResNet最一開始的區塊，並且完成殘差部分，當步長!=1或者輸入、輸出通道不一致時，用1*1的卷積做調整。順序如同註解所以forward就順著下去。

```
class ConvBlock(nn.Module):#U-Net的convolution block
    def __init__(self, in_channels, out_channels):
        super(ConvBlock, self).__init__()
        self.conv_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),
        )

    def forward(self, x):
        return self.conv_block(x)
```

這邊和U-Net的Convolution一樣，因為是做decoder，所以是負責上採樣部分。程式碼同先前U-Net的DoubleConv function。

```

class ResNet34_Unet(nn.Module):
    def __init__(self):
        super(ResNet34_Unet, self).__init__()
        self.initial_layer = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        )
        self.encoder1 = self._make_layer(64, 64, 3, stride=1)
        self.encoder2 = self._make_layer(64, 128, 4, stride=2)
        self.encoder3 = self._make_layer(128, 256, 6, stride=2)
        self.encoder4 = self._make_layer(256, 512, 3, stride=2)

```

這邊按照下表進行建立encoder部分，而`_make_layer` function在這邊先插播以方便表示(即第一個encoder input、output皆是64，然後有3個，第二個input為64、output為128，然後有4個。因為調整輸出通道*2導致圖片size/2，所以步長後面都是2，依此類推。)

```

def _make_layer(self, in_channels, out_channels, blocks, stride):
    layers = [BasicBlock(in_channels, out_channels, stride)]
    for _ in range(1, blocks):#3,4,6,3
        layers.append(BasicBlock(out_channels, out_channels))#把block加進去
    return nn.Sequential(*layers)#將list中的元素展開成參數
#相當於layers = [BasicBlock(in_channels, out_channels, stride)]+[BasicBlock(out_channels, out_channels)]*blocks

```

ResNet34

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
				3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Reference: <https://arxiv.org/abs/1512.03385>

接著呈現initial的其他部分。

```

#Bottleneck部分 512->256
self.bottleneck = nn.Sequential(
    nn.Conv2d(512, 256, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(inplace=True)
)
self.upconv_blocks = nn.ModuleList([
    nn.ConvTranspose2d(768, 768, kernel_size=2, stride=2), #512+256
    nn.ConvTranspose2d(288, 288, kernel_size=2, stride=2), #256+32#原架構圖有誤 不應該是512+32
    nn.ConvTranspose2d(160, 160, kernel_size=2, stride=2), #128+32
    nn.ConvTranspose2d(96, 96, kernel_size=2, stride=2) #64+32
])
self.conv_blocks = nn.ModuleList([
    ConvBlock(768, 32), #藍加灰->藍
    ConvBlock(288, 32),
    ConvBlock(160, 32),
    ConvBlock(96, 32),
])
#最後一個上採樣 32->32
self.final_upconv = nn.ConvTranspose2d(32, 32, kernel_size=2, stride=2)
self.final_conv = nn.Conv2d(32, 1, kernel_size=1)

```

```

def forward(self, x):
    x = self.initial_layer(x)
    #encoder部分
    encoder1_out = self.encoder1(x)
    encoder2_out = self.encoder2(encoder1_out)
    encoder3_out = self.encoder3(encoder2_out)
    encoder4_out = self.encoder4(encoder3_out)
    #bottleneck部分
    x = self.bottleneck(encoder4_out)
    #decoder部分包含上採樣、連接
    x = self.upconv_blocks[0](torch.cat([x, encoder4_out], dim=1))
    x = self.conv_blocks[0](x)
    x = self.upconv_blocks[1](torch.cat([x, encoder3_out], dim=1))
    x = self.conv_blocks[1](x)
    x = self.upconv_blocks[2](torch.cat([x, encoder2_out], dim=1))
    x = self.conv_blocks[2](x)
    x = self.upconv_blocks[3](torch.cat([x, encoder1_out], dim=1))
    x = self.conv_blocks[3](x)
    x = self.final_upconv(x)
    x = self.final_conv(x)

    return F.sigmoid(x)

```

本圖為BottleNeck和UpSampling以及對應的forward部分，因為瓶頸層是512->256且保持輸入輸出大小一致，故要做padding。接下來上採樣在圖片大小部分使用逆卷積將圖片逐步放大回原本大小(皆放大2倍)並且保持輸入輸出一致。並且在最後一次上採樣把圖片尺寸大小放大回原本大小。

2. Data Preprocessing (20%)

A. How you preprocessed your data?

使用data augmentation對訓練集的照片進行鏡射、增加亮度，並且設定(古典機率約30%)的照片做鏡射或是乘以1.2倍之亮度。並傳入模型進行訓練，目的是為了增加模型學習到不同特徵的機會並減少過擬合之情形。此外，僅在訓練集的地方進行資料擴增，而不會在驗證集、測試集做使用。

```
def augment_sample(self, sample):
    image, mask = sample['image'], sample['mask']
    img = Image.fromarray(image)
    # 產生一個隨機數來決定執行哪種擴增
    rand_num = np.random.rand()

    # 鏡射
    if 0 <= rand_num < 0.15:
        img = img.transpose(Image.FLIP_LEFT_RIGHT)
        mask = mask[:, ::-1] # 對遮罩進行相同的鏡射操作

    # 增加亮度
    elif 0.15 <= rand_num < 0.3:
        enhancer = ImageEnhance.Brightness(img)
        img = enhancer.enhance(1.2) # 增加20%的亮度

    sample['image'] = np.array(img)
    sample['mask'] = mask
    return sample
```

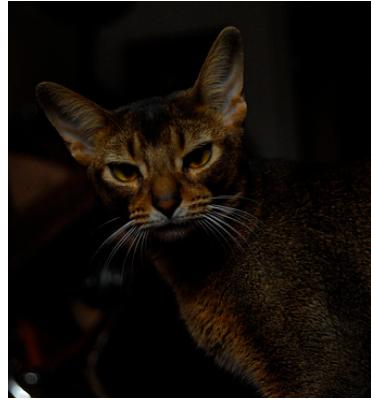
A. What makes your method unique?

(1)方法一鏡射:

使用鏡射將照片翻轉，目的是為了讓模型學習不同拍攝角度下，目標物與背景的關係，而不是特定角度。畢竟若是在訓練過程中集中在某些角度時，容易造成換角度之後預測效果較差，所以想說使用鏡射等於是提供了原圖不一樣的角度讓模型進行訓練。

(2)方法二提高亮度:

由於部分照片光度較暗(如圖一)，造成不易辨別是目標(貓或狗)還是背景。因為語義分割具有pixel-wise之特性，當無法有效區分不同類別間的輪廓時，容易造成訓練效果差。故若是提升亮度，可能較有辦法辨別目標物與背景的差別。當然考量並非所有圖都是偏暗，所以將照片提高亮度20%以避免當遇到拍攝為正常光線時，造成過曝影響模型訓練。



圖一：原圖



圖二：提高亮度後的圖

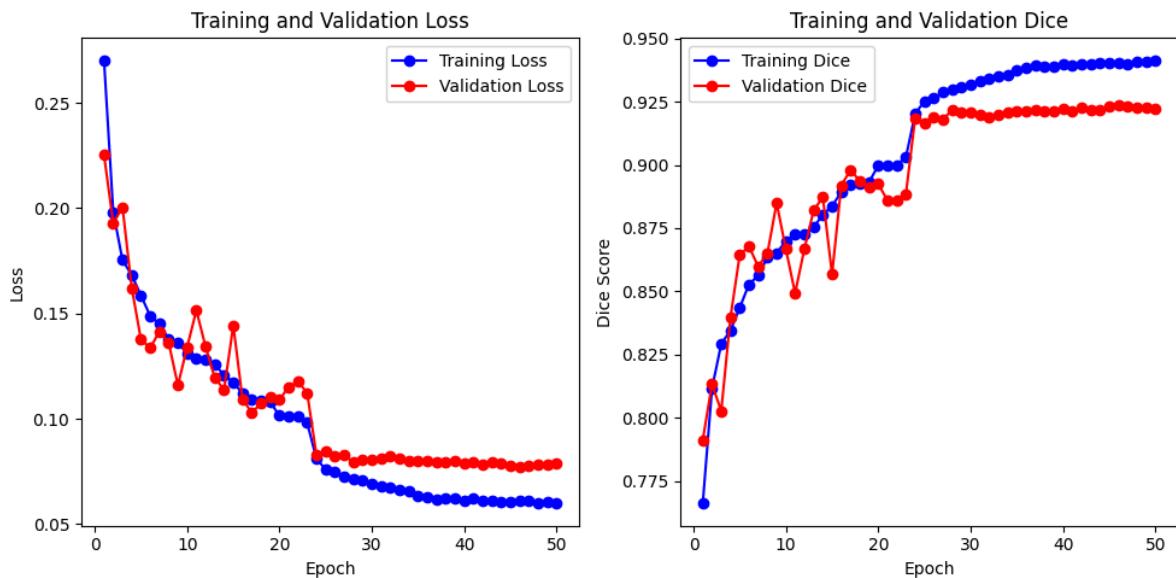
2. Analyze on the experiment results (20%)

A. What did you explore during the training process?

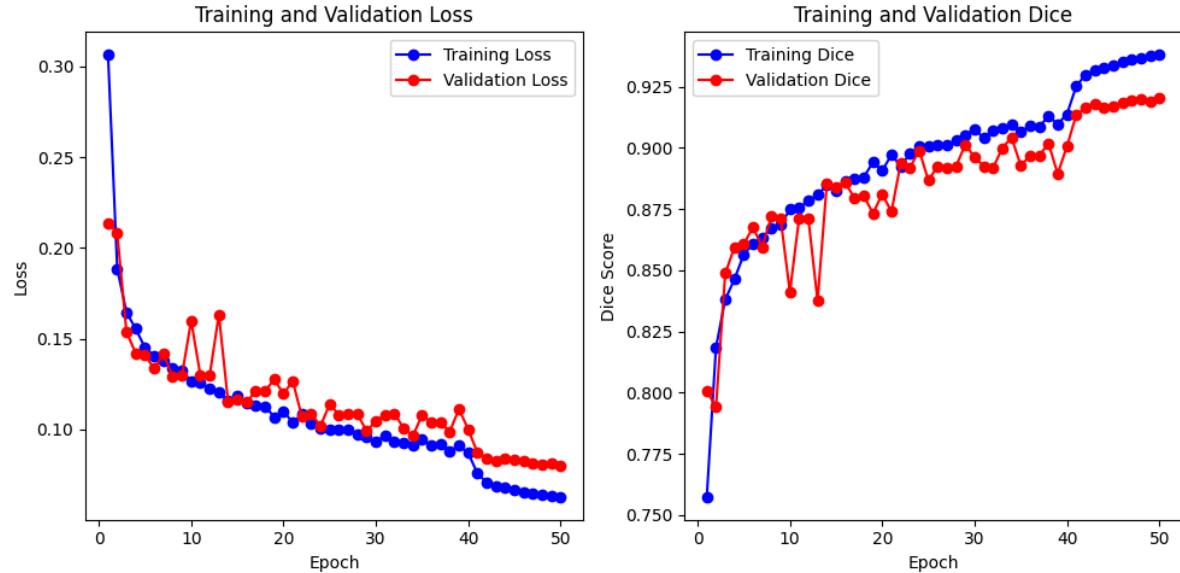
(1) 實驗設置：

本次實驗因設備(GTX1650)因素，故使用 `img size = 256 * 256`, `batch size = 4`, `lr = 0.0001`, `weight_decay`為`0.0005`。在訓練過程中，設定當連續5個epoch中 `valid_loss`未低於最低 `valid_loss`時會將 `lr`乘以 `0.1`，若是超過連續10個epoch的 `valid_loss`未低於最低 `valid_loss`時，則會觸發 `early stopping` 以避免 `overfitting`。

(2) U-Net訓練過程中，Training Loss大致上會穩定下降，Validation Loss在某些 Epoch中會回跳，此外，本次訓練在約第22個epoch時因為連續五個epoch中 `valid_loss` 高於當前最低值所以調降 `lr`，並在第23個epoch時成功降低其 loss value，並且在之後的 epoch 中仍有微幅下降之趨勢(並無在50個epoch前 Early Stopping)。



(3) ResNet34+U-Net訓練過程中，Training Loss也是大致上會穩定下降，Validation Loss在某些Epoch中會回跳，其中在約第30個epoch時大Valid Dice大約達到0.9，且經過大約第42個Epoch時Training、Valid Loss皆再度下降並有持續下降之趨勢，故暫時並無Overfitting的問題。



A. Found any characteristics of the data?

本次資料集共有12種貓、25種狗，但因為是判斷是否為貓和狗以及背景，故類別不平衡之問題在本lab較不存在，此外有發現部分相片較為昏暗，故在資料擴增部分有隨機調整亮度以求能夠解決當光線昏暗時較難分辨邊界之情況。

3. Execution command (0%)

A. The command and parameters for the training process

U-Net腳本：

```
python train.py --model unet --data_path C:/Users/蘇柏叡
/Desktop/DL_Lab3_313553024_蘇柏叡/dataset --epochs 50 --batch_size 4 --gpu
0 --learning-rate 0.0001
```

ResNet34腳本：

```
python train.py --model resnet34_unet --data_path C:/Users/蘇柏叡
/Desktop/DL_Lab3_313553024_蘇柏叡/dataset --epochs 50 --batch_size 4 --gpu
0 --learning-rate 0.0001
```

以下為command說明

```
def get_args():
    parser = argparse.ArgumentParser(description='Train the UNet on images and target masks')
    parser.add_argument("-g", "--gpu", help="gpu id", default=0, type=int)
    #parser.add_argument("-n", "--name", help="Experiment name", default="", type=str)
    parser.add_argument('--model', '-m', type=str, default='unet', help='target model')
    parser.add_argument('--data_path', '-path', type=str, default='C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/dataset', help='dataset path')
    parser.add_argument('--epochs', '-e', type=int, default=50, help='number of epochs')
    parser.add_argument('--batch_size', '-b', type=int, default=4, help='batch size')
    parser.add_argument('--learning_rate', '-lr', type=float, default=1e-4, dest='lr', help='learning rate')
    return parser.parse_args()
```

--data_path 資料集的位置

在我的電腦上是C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/dataset，此外若原本為空的話，會先下載。

--batch_size 可以決定一次批量大小，因設備關係選擇4

--gpu 選擇你要的gpu

--model 選擇想要的模型可以鍵入 resnet34_unet、unet其中一種

--epochs 輸入要訓練的周期數

--learning-rate 輸入學習率

A. The command and parameters for the inference process

U-Net腳本：

```
python inference.py --model unet --weights C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/saved_models/DL_Lab3_UNet_313553024_蘇柏叡.pth --data_path C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/dataset --batch_size 8 --gpu 0
```

ResNet34腳本：

```
python inference.py --model resnet34_unet --weights C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/saved_models/DL_Lab3_ResNet34_UNet_313553024_蘇柏叡.pth --data_path C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/dataset --batch_size 8 --gpu 0
```

以下為command說明

```
import argparse
import torch
from utils import dice_score
from oxford_pet import load_dataset
from models.resnet34_unet import ResNet34_Unet
from models.unet import UNet
from torch.utils.data import DataLoader

def get_args():
    parser = argparse.ArgumentParser(description='Predict masks from input images')
    parser.add_argument('--model', default='resnet34', help='model type: resnet34 or unet')
    parser.add_argument('--weights', default='saved_models/best.pth', help='path to stored model weights')
    parser.add_argument('--data_path', type=str, required=True, help='path to the input data')
    parser.add_argument('--batch_size', '-b', type=int, default=8, help='batch size')
    parser.add_argument("-g", "--gpu", help="gpu id", default=0, type=int)
    return parser.parse_args()
```

--model 選擇想要的模型可以鍵入 resnet34_unet、unet其中一種

--weights 選擇要的權重檔案

Resnet34_unet權重檔案為：

C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡
/saved_models/DL_Lab3_ResNet34_UNet_313553024_蘇柏叡. pth

unet權重檔案為：

C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡
/saved_models/DL_Lab3_UNet_313553024_蘇柏叡. pth

--data_path 資料集的位置

在我的電腦上是C:/Users/蘇柏叡/Desktop/DL_Lab3_313553024_蘇柏叡/dataset，此外若原本為空的話，會先下載後再跑inference。

--batch_size 可以決定一次批量大小，因設備關係選擇8

--gpu 選擇你要的gpu

4. Discussion (20%)

A. What architecture may bring better results?

U-Net: 0.9260

```
_Lab3_313553024_蘇柏叢\dataset --batch_size 8 --gpu 0      python inference.py  
--model unet --weights saved_models/best_unet_aug.pth --data_path C:/Users//  
蘇柏叢/Desktop/DLP_Lab3_313553024_蘇柏叢\dataset --batch_size 8 --gpu 0  
unet Testing Dice Score:0.9260
```

ResNet34+U-Net: 0.9257

```
PS C:\Users\蘇柏叢\Desktop\Lab3_313553024_蘇柏叢\src> python inference.  
_unet --weights saved_models/best.pth --data_path C:/Users/蘇柏叢/Desktop/DLP  
_Lab3_313553024_蘇柏叢\dataset --batch_size 8 --gpu 0  
resnet34_unet Testing Dice Score:0.9257
```

以這兩個model來看U-Net的Dice score略高一些(0.003)，且以訓練週期中模型穩定度亦是U-Net較為穩定一些(因為ResNet34+U-Net)較易有起伏。不過題外話，以訓練時長與資源來說，U-Net平均一個Epoch需耗時約450秒，而ResNet34+U-Net則是約140秒，且U-Net較吃資源，所以若是以資源與設備有限來探討的話，ResNet+U-Net會較為合適。

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

因為本Lab是分辨target(貓或狗)和background的binary semantic segmentation，若是要區分12種品種貓、25種品種的狗multi-classes的semantic segmentation則是較為浩大的工程，就不會是單純區分是貓或狗或是背景而已。

此外，也有在網路上看到有使用遷移學習應用在本資料集，或許也可以用來比較和從頭開始建model的方法進行比較。