國立陽明交通大學 深度學習實驗 LAB3

313553036 多工所 1 葉軒宇

1. Overview of your lab 3

這次實驗室在實作 UNet、ResNet34+UNet 兩個 Binary Semantic Segmentation 模型，並且針對 Oxford-IIIt pet 的 dataset 去作訓練。希望可以在每張圖片上區別前景以及後景的部分。

2. Implementation Details

A. Details of your training, evaluating, inferencing code

A.1 training

一開始先定義損失函數以及優化器的部分，然後初始化 dice。接下來在每次 epoch 中在驗證集上評估分數，然後紀錄最好的 dice 分數，並且在訓練過程中一邊繪製學習曲線。

```python
def train_model(model, train_loader, val_loader, num_epochs=50, learning_rate=1e-4, logger=None):
    device = get_device()
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    best_dice_score = 0.0
    train_dice_history = []
    val_dice_history = []

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0
        train_dice = 0.0

        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs}")
        for images, masks in progress_bar:
            images, masks = images.to(device), masks.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, masks)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            pred = torch.argmax(outputs, dim=1)
            train_dice += dice_score(pred, masks)

            progress_bar.set_postfix({'loss': f'{loss.item():.4f}'})

        train_loss /= len(train_loader)
        train_dice /= len(train_loader)
```

```python
        val_loss, val_dice = evaluate_model(model, val_loader, criterion, device)

        train_dice_history.append(train_dice.cpu())
        val_dice_history.append(val_dice.cpu())

        if logger:
            logger.info(f"Epoch {epoch+1}/{num_epochs}")
            logger.info(f"Train Loss: {train_loss:.4f}, Train Dice: {train_dice:.4f}")
            logger.info(f"Val Loss: {val_loss:.4f}, Val Dice: {val_dice:.4f}")

        if val_dice > best_dice_score:
            best_dice_score = val_dice
            save_checkpoint( state: {
                'epoch': epoch + 1,
                'state_dict': model.state_dict(),
                'optimizer': optimizer.state_dict(),
                'best_dice_score': best_dice_score,
            }, filename=f"../saved_models/best_model_{type(model).__name__}.pth")
            if logger:
                logger.info(f"New best model saved with Dice score: {best_dice_score:.4f}")

    return model, train_dice_history, val_dice_history
```

```python
def evaluate_model(model, data_loader, criterion, device):
    model.eval()
    val_loss = 0.0
    val_dice = 0.0

    with torch.no_grad():
        for images, masks in data_loader:
            images, masks = images.to(device), masks.to(device)
            outputs = model(images)
            loss = criterion(outputs, masks)
            val_loss += loss.item()
            pred = torch.argmax(outputs, dim=1)
            val_dice += dice_score(pred, masks)

    val_loss /= len(data_loader)
    val_dice /= len(data_loader)
    return val_loss, val_dice
```

最後根據要訓練的不同模型去調整想要的參數

```python
if __name__ == "__main__":
    data_path = "../dataset"
    batch_size = 2
    train_loader = load_dataset(data_path,  mode: "train", batch_size=batch_size)
    val_loader = load_dataset(data_path,  mode: "valid", batch_size=batch_size)


    log_dir = "../logs"
    os.makedirs(log_dir, exist_ok=True)
    logger = setup_logger( name: 'train_log',  log_file: f'{log_dir}/train.log')


    device = get_device()
    logger.info(f"Using device: {device}")

    # training UNet
    logger.info("Starting UNet training")
    unet_model = UNet(n_channels=3, n_classes=2)
    unet_model, unet_train_history, unet_val_history = train_model(unet_model, train_loader, val_loader, num_epochs=50, learning_rate=0.0006,
    plot_learning_curve(unet_train_history, unet_val_history,  save_path: f'{log_dir}/unet_learning_curve.png')

    # training resnet34_unet
    logger.info("Starting ResNet34-UNet training")
    resnet_unet_model = UNetResNet34(num_classes=2)
    resnet_unet_model, resnet_train_history, resnet_val_history = train_model(resnet_unet_model, train_loader, val_loader, num_epochs=50, le
    plot_learning_curve(resnet_train_history, resnet_val_history,  save_path: f'{log_dir}/resnet_unet_learning_curve.png')

    logger.info("Training completed")
```

```python
class Model:
    def __init__(self, input_size = 2, hidden_size = 10, lr = 0.1):
        self.layer1 = Layer(input_size, hidden_size)
        self.layer2 = Layer(hidden_size, hidden_size)
        self.output = Layer(hidden_size, 1)
        self.lr = lr
        self.loss = []
        self.epoch = 0

    def train(self, x, y, epoch):
        self.epoch = epoch
        for i in range(epoch):
            output = self.output.forward(self.layer2.forward(self.layer1.forward(x)))
            loss, grad = MSELoss(output, y)
            self.layer1.backward(self.layer2.backward(self.output.backward(grad, self.lr), self.lr), self.lr)
            self.loss.append(loss)
            if i % 5000 == 0: #control how many cycle print loss
                print(f"epoch {i}  loss : {loss}")
        self.prediction = output
        plt.subplot(2, 1, 1)
        plt.title("Learning Curve", fontsize=18)
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.plot(self.loss)
        return output

    def show_result(self, x, y):
        print(f"Accuracy : {sum((self.prediction > 0.5) == (y == 1)) / y.size}")
        print("Prediction : ")
        for i in range(y.size):
            print(f"Iter{i + 1} |    Ground truth: {y[i]} |     prediction: {self.prediction[i]} |")
        show_result(x, y, self.prediction > 0.5)

plt.figure(figsize=(12, 12))
```

A.2 evaluating

分別載入 UNet 還有 ResNet34_UNet 的訓練結果，然後使用 test 的訓練集來評估兩種模型的 dice_score。

```python
def evaluate_model(model, data_loader, device):
    model.eval()
    total_dice = 0.0

    with torch.no_grad():
        for images, masks in tqdm(data_loader, desc="Evaluating"):
            images, masks = images.to(device), masks.to(device)
            outputs = model(images)
            pred = torch.argmax(outputs, dim=1)
            total_dice += dice_score(pred, masks)

    avg_dice = total_dice / len(data_loader)
    return avg_dice
```

```python
if __name__ == "__main__":
    data_path = "../dataset"
    test_loader = load_dataset(data_path, "test")
    device = get_device()


    unet_model = UNet(n_channels=3, n_classes=2)
    _, _ = load_checkpoint("../saved_models/best_model_UNet.pth", unet_model)
    unet_model = unet_model.to(device)
    unet_dice = evaluate_model(unet_model, test_loader, device)

    print(f"UNet Dice Score on Test Set: {unet_dice:.4f}")

    resnet_unet_model = UNetResNet34(num_classes=2)
    _, _ = load_checkpoint("../saved_models/best_model_UNetResNet34.pth", resnet_unet_model)
    resnet_unet_model = resnet_unet_model.to(device)
    resnet_unet_dice = evaluate_model(resnet_unet_model, test_loader, device)
    print(f"ResNet34-UNet Dice Score on Test Set: {resnet_unet_dice:.4f}")
```

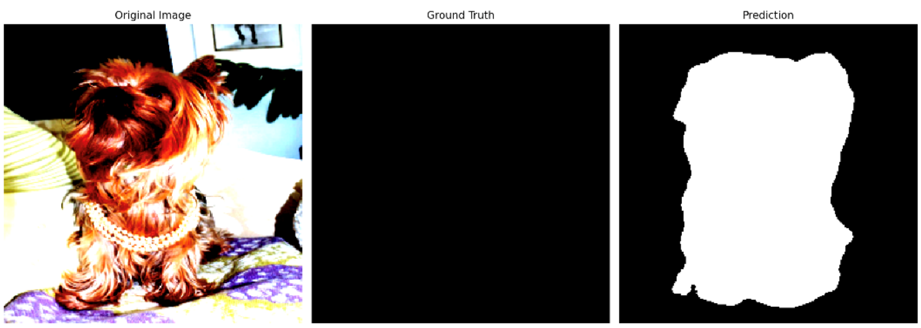
A.3 inferencing code

這部分是根據單張圖片來進行推理，並且將結果視覺化。

```python
def preprocess_image(image_path):
    image = Image.open(image_path).convert("RGB")
    transform = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    return transform(image).unsqueeze(0)


2 usages
def inference(model, image_path, device):
    model.eval()
    image = preprocess_image(image_path).to(device)

    with torch.no_grad():
        output = model(image)
        prediction = torch.argmax(output, dim=1)

    return image.squeeze().cpu(), prediction.squeeze().cpu()
```

```python
if __name__ == "__main__":
    device = get_device()

    unet_model = UNet(n_channels=3, n_classes=2)
    _, _ = load_checkpoint( filename: "../saved_models/best_model_UNet.pth", unet_model)
    unet_model = unet_model.to(device)

    resnet_unet_model = UNetResNet34(num_classes=2)
    _, _ = load_checkpoint( filename: "../saved_models/best_model_UNetResNet34.pth", resnet_unet_model)
    resnet_unet_model = resnet_unet_model.to(device)

    image_path = "../dataset/oxford-iiit-pet/images/yorkshire_terrier_102.jpg"

    unet_image, unet_prediction = inference(unet_model, image_path, device)
    resnet_unet_image, resnet_unet_prediction = inference(resnet_unet_model, image_path, device)

    visualize_prediction(unet_image, torch.zeros_like(unet_prediction), unet_prediction)
    visualize_prediction(resnet_unet_image, torch.zeros_like(resnet_unet_prediction), resnet_unet_prediction)
```

結果會像這樣將前景與後景區分出來



B. Details of your model(UNet & ResNet34_UNet)

B.1 UNet

UNet 無論是在前或著後半段的步驟，都包含一個雙層的 convolution，再根據需要 maxpool 或著 concatenation 去作調整。在 upsample 的時候，因為會發生大小不同的情況，所以根據兩者間的差異進行 padding。

```python
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

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

```python
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes):
        super().__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes

        self.inc = DoubleConv(n_channels, out_channels: 64)
        self.down1 = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv( in_channels: 64, out_channels: 128)
        )
        self.down2 = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv( in_channels: 128, out_channels: 256)
        )
        self.down3 = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv( in_channels: 256, out_channels: 512)
        )
        self.down4 = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv( in_channels: 512, out_channels: 1024)
        )
        self.up1 = DoubleConv(1024 + 512, out_channels: 512)
        self.up2 = DoubleConv(512 + 256, out_channels: 256)
        self.up3 = DoubleConv(256 + 128, out_channels: 128)
        self.up4 = DoubleConv(128 + 64, out_channels: 64)
        self.outc = nn.Conv2d( in_channels: 64, n_classes, kernel_size=1)
```

```python
    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)

        x = self._up_and_concat(x5, x4, self.up1)
        x = self._up_and_concat(x, x3, self.up2)
        x = self._up_and_concat(x, x2, self.up3)
        x = self._up_and_concat(x, x1, self.up4)

        logits = self.outc(x)
        return logits

    def _up_and_concat(self, x1, x2, up_layer):
        x1 = F.interpolate(x1, scale_factor=2, mode='bilinear', align_corners=True)

        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]
        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])

        x = torch.cat([x2, x1], dim=1)
        return up_layer(x)
```

B.2 ResNet34+UNet

這個模型主要就是分為兩個部分，前面 ResNet 的部分大概就是不停地進行 conv+BN+ReLU 這個步驟，所以我將前半部分寫成一個大的 block。

```python
class BasicBlock(nn.Module):
    expansion = 1

    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)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != self.expansion * out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, self.expansion * out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion * out_channels)
            )

    def forward(self, x):
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = self.relu(out)
        return out
```

```python
class ResNet34Encoder(nn.Module):
    def __init__(self, block, layers):
        super(ResNet34Encoder, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d( in_channels: 3,  out_channels: 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block,  out_channels: 64, layers[0])
        self.layer2 = self._make_layer(block,  out_channels: 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block,  out_channels: 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block,  out_channels: 512, layers[3], stride=2)

    4 usages
    def _make_layer(self, block, out_channels, blocks, stride=1):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride))
        self.in_channels = out_channels * block.expansion
        for _ in range(1, blocks):
            layers.append(block(self.in_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        features = []
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        features.append(x)
        x = self.maxpool(x)
        x = self.layer1(x)
        features.append(x)
        x = self.layer2(x)
        features.append(x)
        x = self.layer3(x)
        features.append(x)
        x = self.layer4(x)
        features.append(x)
        return features
```

後半部分則是重複 conv+concatenation，一樣須注意兩者間的大小是否有差異，將整體完成後即可。

```python
class DecoderBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DecoderBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

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

```python
class UNetResNet34(nn.Module):
    def __init__(self, num_classes=2):
        super(UNetResNet34, self).__init__()
        self.encoder = ResNet34Encoder(BasicBlock, layers: [3, 4, 6, 3])
        self.decoder5 = DecoderBlock( in_channels: 512,  out_channels: 512)
        self.decoder4 = DecoderBlock(512 + 256,  out_channels: 256)
        self.decoder3 = DecoderBlock(256 + 128,  out_channels: 128)
        self.decoder2 = DecoderBlock(128 + 64,  out_channels: 64)
        self.decoder1 = DecoderBlock(64 + 64,  out_channels: 32)
        self.final_conv = nn.Conv2d( in_channels: 32, num_classes, kernel_size=1)

    def forward(self, x):
        features = self.encoder(x)

        x = features[-1]
        x = self.decoder5(x)
        x = self._up_and_concat(x, features[-2], self.decoder4)
        x = self._up_and_concat(x, features[-3], self.decoder3)
        x = self._up_and_concat(x, features[-4], self.decoder2)
        x = self._up_and_concat(x, features[-5], self.decoder1)
        x = F.interpolate(x, size=(256, 256), mode='bilinear', align_corners=True)
        x = self.final_conv(x)

        return x

    4 usages
    def _up_and_concat(self, x1, x2, up_layer):
        x1 = F.interpolate(x1, scale_factor=2, mode='bilinear', align_corners=True)

        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])

        x = torch.cat( tensors: [x2, x1], dim=1)
        return up_layer(x)
```

3. Data preprocessing

A. How you preprocessed your data

除了原本作業規定就有的 resize 成(256*256)外，還使用了隨機抖動，其中包刮亮度、對比度、飽和度。除此之外，還使用了 ImageNet 的均值以及標準差進行標準化。

```python
def get_transform(mode):
    if mode == "train":
        return transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])
    else:
        return transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
        ])


5 usages
def load_dataset(data_path, mode, batch_size=2):
    transform = get_transform(mode)
    dataset = OxfordPetDataset(root=data_path, mode=mode, transform=transform)
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=(mode == "train"))
    return dataloader
```

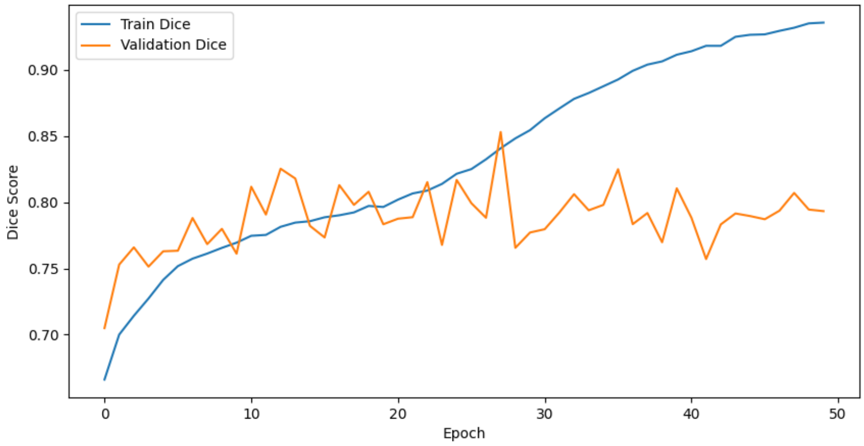
B. What makes your method unique?

Color Jitter 可以在圖片特別明亮或是特別昏暗的時候幫助分割場，還可以幫助模型學會在不同變化下的準確率。除此之外還用了 ImageNet 統計出來的資料，讓在不切割、調整圖片大小的情況下，提升圖片的多樣性，進而提升泛化能力。

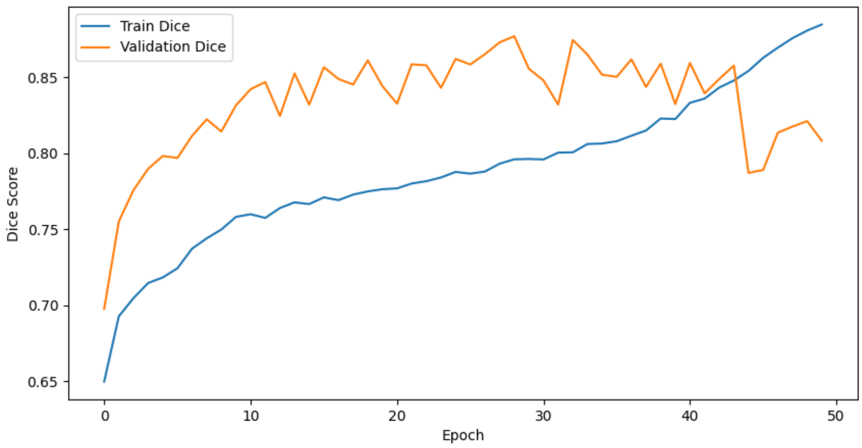
4. Analyze on the experiment results

A. What did you explore during the training process

根據訓練的過程可以發現，因為 UNet 使用了大量的 convolution，因此在訓練時間上相比 Resnet34+UNet 會多上不少。即使因為訓練時間真的太長，在 epoch_number 我只設定 50，但由圖形可以看出來，UNet 在 epoch 超過 40 之後，train dice 的分數提升反而是開始加速的。而使用 ResNet+UNet 的模型則是在前期就有比較好的分數。



ResNet34+UNet



UNet

B. Found any characteristic of the data?

因為照片真的非常多，甚至仔細去看一張張照片的話，有些狗的毛長到以為那是一根拖把，跟長得像袋鼠的貓。這樣可以讓模型更好的去學習，對之後的泛化也可以有更好的效果。

5. Execution command

A. The command and parameters for the training process

除了設定 device 非常重要，因為會直接影響有沒有使用到 gpu 外。Batch size 是一個非常重要的點，一開始我跟前幾次 lab 一樣都條 32，結果 cpu、gpu 使用率都超級差，調成 4 之後才變好。

Model:	UNet/ResNet34_UNet
Epoch:	50/50
Batch_size:	4/4
Learning_rate:	0.0006/0.001

B. The command and parameters for the inference process

Image_path:
"../dataset/oxford-iiit-pet/images/yorkshire_terrier_102.jpg"

6. Dicussion

A. What architecture may bring better results

這很難有一個直接的答案，根據你對圖片預處理的不同，以及資料大小的不同，兩個模型上會各有優劣。但就我的實驗結果而言，在相同的 epoch 數下，UNet 得到的 dice_score 是較高的。

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

一個最直觀可以想到的就是根據不同的圖片預處理，可以帶來怎麼樣的差別，除此之外就是根據驗證、訓練集的大小來看對兩個模型在 dice_score 得分上的差異。

7. Cite
https://blog.csdn.net/weixin_43977304/article/details/121497425

UNet paper
[1505.04597v1] U-Net: Convolutional Networks for Biomedical Image Segmentation (arxiv.org)

ResNet34_UNet paper
(PDF) Deep learning-based pelvic levator hiatus segmentation from ultrasound images (researchgate.net)

Anthropic. (2023). Claude AI [Computer software]. https://www.anthropic.com or https://www.anthropic.ai