Lab3: Binary Semantic Segmentation

312551133

鄧長軒

1. Overview of your lab 3

在Lab3中，我實作了UNet模型與ResNet34_UNet模型，並從頭開始訓練它們，將參數調整至適合本次Lab的輸入圖片，以及選擇合理的torch api，學習到encoder與decoder的概念，concatenation的目的，最後在test資料集獲得0.88與0.89的分數。

2. Implementation Details

Details of training code

```
model.to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=args.learning_rate, momentum=0.9)

train_scores = []
valid_scores = []
for epoch in range(args.epochs):
    dataset = oxford_pet.load_dataset(os.path.join(args.data_path, 'dataset', 'oxford-iiit-pet'), mode='train')
    dataloader = DataLoader(dataset, batch_size=args.batch_size, shuffle=True)
    for sample in tqdm(dataloader, desc='train', leave=False):
        input = sample["image"]
        input = input.float().to(device)
        label = sample["mask"].to(device)
        optimizer.zero_grad()
        output = model(input)
        loss = nn.MSELoss()(output, label)
        loss.backward()
        optimizer.step()
    train_score=evaluate.evaluate(model, args, 'train', device)
    valid_score=evaluate.evaluate(model, args, 'valid', device)
    train_scores.append(train_score)
    valid_scores.append(valid_score)
    print(f'Epoch [{epoch+1}/{args.epochs}], train dice score: {train_score:.4f}, valid dice score: {valid_score:.4f}')
```

Details of training code

- 設定optimizer為SGD。
- train_scores與valid_scores儲存每個epoch的score。
- 將sample[“image”]與sample[“mask”]移至GPU。
- Loss是選nn.MSELoss()，因為我的模型輸出的值是介於0~1之間，小於0.5則代表label=0，大於0.5為label=1。如果用CrossEntropy，它會用softmax函式，使全部輸出值(256*256個像素)相加等於1，那每個值都會小於0.5，全部預測為0，這不是我想要的結果。因此我保留輸出值，用MSE計算loss。
- 用evaluate函式計算dice score，輸出此epoch的score。

Details of evaluating code

```
def evaluate(model, args, mode, device):
    dataset = oxford_pet.load_dataset(os.path.join(args.data_path, 'dataset', 'oxford-iiit-pet'), mode=mode)
    dataloader = DataLoader(dataset, batch_size=args.batch_size)
    model.eval()
    score = 0
    number = 0
    for sample in tqdm(dataloader, desc='eval-'+mode, leave=False):
        input = sample["image"]
        input = input.float().to(device)
        label = sample["mask"].to(device)
        outputs = model(input)
        outputs = torch.where(outputs < 0.5, torch.tensor(0., device=device), torch.tensor(1., device=device))
        score += utils.dice_score(outputs, label)
        number += 1
    return (score/number)
```

Details of evaluating code

- 傳入的參數mode包括train, valid, test。
- score儲存全部dice score的總和，number代表全部圖片的數量。
- 由於model的預測結果output裡的值介於0~1，因此透過torch.where將小於0.5的值變成0，大於0.5的值變成1。
- 然後用utils.dice_score計算預測結果與ground true的dice score，將分數加到score裡。
- 最後回傳平均dice score。

Details of inferencing code

```python
if __name__ == '__main__':
    args = get_args()
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    dataset = oxford_pet.load_dataset(os.path.join(args.data_path, 'dataset', 'oxford-iiit-pet'), mode='test')
    dataloader = DataLoader(dataset, batch_size=args.batch_size)
    model = torch.load(os.path.join(args.data_path, 'saved_models', args.model), map_location=device)
    model.eval()
    score = 0
    number = 0
    for sample in tqdm(dataloader, desc='inference', leave=False):
        input = sample["image"]
        input = input.float().to(device)
        label = sample["mask"].to(device)
        outputs = model(input)
        outputs = torch.where(outputs < 0.5, torch.tensor(0., device=device), torch.tensor(1., device=device))
        score += utils.dice_score(outputs, label)
        number += 1
    print(f'Dice score: {score/number:.4f}')
```

Details of inferencing code

inferencing code與evaluating code幾乎一樣，差別在於：

1. inference時，dataset的mode限定是test。
2. 可以透過terminal打指令接收參數。
3. 最後是印出分數。

Details of model UNet

Block將兩層Conv串在一起，可透過參數in_channels與out_channels決定要將channels變成多少。

不改變尺寸大小，所以kernel_size為3，stride為1，padding為1。

```
class Block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Block, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )
    def forward(self, x):
        x = self.block(x)
        return x
```

Details of model UNet

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

        self.encoder1 = Block(3, 64)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder2 = Block(64, 128)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder3 = Block(128, 256)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.encoder4 = Block(256, 512)
        self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.bottleneck = Block(512, 1024)

        self.upconv4 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
        self.decoder4 = Block(1024, 512)
        self.upconv3 = nn.ConvTranspose2d(512, 256, kernel_size=2, stride=2)
        self.decoder3 = Block(512, 256)
        self.upconv2 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2)
        self.decoder2 = Block(256, 128)
        self.upconv1 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2)
        self.decoder1 = Block(128, 64)

        self.outconv = nn.Conv2d(64, 1, kernel_size=1)
        self.bn = nn.BatchNorm2d(1)
        self.sigmoid = nn.Sigmoid()
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
```

Details of model UNet

輸入的圖片是3*256*256，因此流程為：
3*256*256→64*128*128→128*64*64→256*32*32→512*16*16。

channels的改變由Block完成，尺寸的變小由nn.MaxPool2d完成，尺寸的變大由nn.ConvTranspose2d完成。

最後將channels變成1，標準化，Sigmoid將輸出變成0~1。

Conv2d的weight初始化用nn.init.kaiming_normal_，BatchNorm2d的weight初始化成常數。

Details of model UNet

```
def forward(self, x):
    encoder1 = self.encoder1(x)
    encoder2 = self.encoder2(self.pool1(encoder1))
    encoder3 = self.encoder3(self.pool2(encoder2))
    encoder4 = self.encoder4(self.pool3(encoder3))

    bottleneck = self.bottleneck(self.pool4(encoder4))

    decoder4 = self.upconv4(bottleneck)
    decoder4 = torch.cat((encoder4, decoder4), dim=1)
    decoder4 = self.decoder4(decoder4)
    decoder3 = self.upconv3(decoder4)
    decoder3 = torch.cat((encoder3, decoder3), dim=1)
    decoder3 = self.decoder3(decoder3)
    decoder2 = self.upconv2(decoder3)
    decoder2 = torch.cat((encoder2, decoder2), dim=1)
    decoder2 = self.decoder2(decoder2)
    decoder1 = self.upconv1(decoder2)
    decoder1 = torch.cat((encoder1, decoder1), dim=1)
    decoder1 = self.decoder1(decoder1)

    x = self.outconv(decoder1)
    x = self.bn(x)
    x = self.sigmoid(x)

    return x
```

Details of model UNet

接著把module串在一起。

UNet的構造有將encoder的各階段concatenation到decoder上，我是用torch.cat接上去，dim=1代表是接在channels的維度上。

由於會需要encoder的各階段結果，所以必須將結果儲存到變數裡，不能用nn.Sequential直接串起來。

Details of model ResNet34_UNet

先簡單介紹3個class：

Bottleneck：是ResNet裡的結構，進行兩層conv2d，且有residual計算。

Block：是UNet裡的結構，單純進行兩層conv2d。

ResNet34_UNet：將所有東西串在一起，包含完整的encoder與

decoder，input為3*256*256的圖片，輸出為1*256*256的label。

Details of model ResNet34_UNet

Bottleneck：參數skip決定是否要進行尺寸縮小，如果是，則第一層conv2d的stride為2，residual也要經過stride=2縮小。

```python
class Bottleneck(nn.Module):
    def __init__(self, in_channels, out_channels, skip):
        super(Bottleneck, self).__init__()

        if skip==1:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

        self.bn = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)

        self.skip = skip
        if skip==1:
            self.skipconv = nn.Conv2d(in_channels, out_channels, kernel_size=(1, 1), stride=(2, 2), padding=(0, 0))
            self.skipbn = nn.BatchNorm2d(out_channels)
```

```python
def forward(self, x):
    residual = x

    x = self.conv1(x)
    x = self.bn(x)
    x = self.relu(x)

    x = self.conv2(x)
    x = self.bn(x)
    x = self.relu(x)

    if self.skip==1:
        residual = self.skipconv(residual)
        residual = self.skipbn(residual)

    x = x + residual
    x = self.relu(x)

    return x
```

Details of model ResNet34_UNet

Block：將兩層conv串在一起，可透過參數in_channels與out_channels決定要將channels變成多少。

不改變大小，所以kernel_size為3，stride為1，padding為1。

```python
class Block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Block, self).__init__()
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )
    def forward(self, x):
        x = self.block(x)
        return x
```

Details of model ResNet34_UNet

ResNet34_UNet：

encoder部分使用ResNet的概念，

進行4層encode，

每一層的第一個conv

會尺寸縮小，channels變大，

而第一層encoder沒有尺寸變化。

```python
class ResNet34_UNet(nn.Module):

    def __init__(self):
        super(ResNet34_UNet, self).__init__()

        self.conv = nn.Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))

        self.encoder1 = nn.Sequential(
            Bottleneck(64, 64, 0),
            Bottleneck(64, 64, 0),
            Bottleneck(64, 64, 0),
        )
        self.encoder2 = nn.Sequential(
            Bottleneck(64, 128, 1),
            Bottleneck(128, 128, 0),
            Bottleneck(128, 128, 0),
            Bottleneck(128, 128, 0),
        )
        self.encoder3 = nn.Sequential(
            Bottleneck(128, 256, 1),
            Bottleneck(256, 256, 0),
            Bottleneck(256, 256, 0),
            Bottleneck(256, 256, 0),
            Bottleneck(256, 256, 0),
            Bottleneck(256, 256, 0),
        )
        self.encoder4 = nn.Sequential(
            Bottleneck(256, 512, 1),
            Bottleneck(512, 512, 0),
            Bottleneck(512, 512, 0),
        )
```

Details of model ResNet34_UNet

decoder部分，

每一個ConvTranspose2d

將channels數變一半，尺寸變大2倍，

再由Block將channels變成32。

最後由outconv將channels變成1。

```
self.bottleneck = Block(512,256)

self.upconv4 = nn.ConvTranspose2d(768, 384, kernel_size=2, stride=2)
self.decoder4 = Block(384, 32)
self.upconv3 = nn.ConvTranspose2d(288, 144, kernel_size=2, stride=2)
self.decoder3 = Block(144, 32)
self.upconv2 = nn.ConvTranspose2d(160, 80, kernel_size=2, stride=2)
self.decoder2 = Block(80, 32)
self.upconv1 = nn.ConvTranspose2d(96, 48, kernel_size=2, stride=2)
self.decoder1 = Block(48, 32)

self.outconv = nn.Sequential(
    nn.ConvTranspose2d(32, 32, kernel_size=2, stride=2),
    Block(32, 32),
    Block(32, 1),
)

self.bn = nn.BatchNorm2d(1)
self.sigmoid = nn.Sigmoid()


for m in self.modules():
    if isinstance(m, nn.Conv2d):
        nn.init.kaiming_normal_(m.weight)
    elif isinstance(m, nn.BatchNorm2d):
        nn.init.constant_(m.weight, 1)
        nn.init.constant_(m.bias, 0)
```

Details of model ResNet34_UNet

forward將所有東西串在一起：

前處理

→經過4層encode

→經過4層concatenation & decode

→輸出channels變1

→經過sigmoid

```python
def forward(self, x):
    x = self.conv(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    encoder1 = self.encoder1(x)
    encoder2 = self.encoder2(encoder1)
    encoder3 = self.encoder3(encoder2)
    encoder4 = self.encoder4(encoder3)

    bottleneck = self.bottleneck(encoder4)

    decoder4 = torch.cat((encoder4, bottleneck), dim=1)
    decoder4 = self.upconv4(decoder4)
    decoder4 = self.decoder4(decoder4)

    decoder3 = torch.cat((encoder3, decoder4), dim=1)
    decoder3 = self.upconv3(decoder3)
    decoder3 = self.decoder3(decoder3)

    decoder2 = torch.cat((encoder2, decoder3), dim=1)
    decoder2 = self.upconv2(decoder2)
    decoder2 = self.decoder2(decoder2)

    decoder1 = torch.cat((encoder1, decoder2), dim=1)
    decoder1 = self.upconv1(decoder1)
    decoder1 = self.decoder1(decoder1)

    x = self.outconv(decoder1)
    x = self.bn(x)
    x = self.sigmoid(x)
    return x
```

3. Data Preprocessing

How you preprocessed your data

我在SimpleOxfordPetDataset裡對sample[“image”]進行處理，
將值除以255，變成介於0~1之間，
再用常見的平均與標準差對圖片標準化。

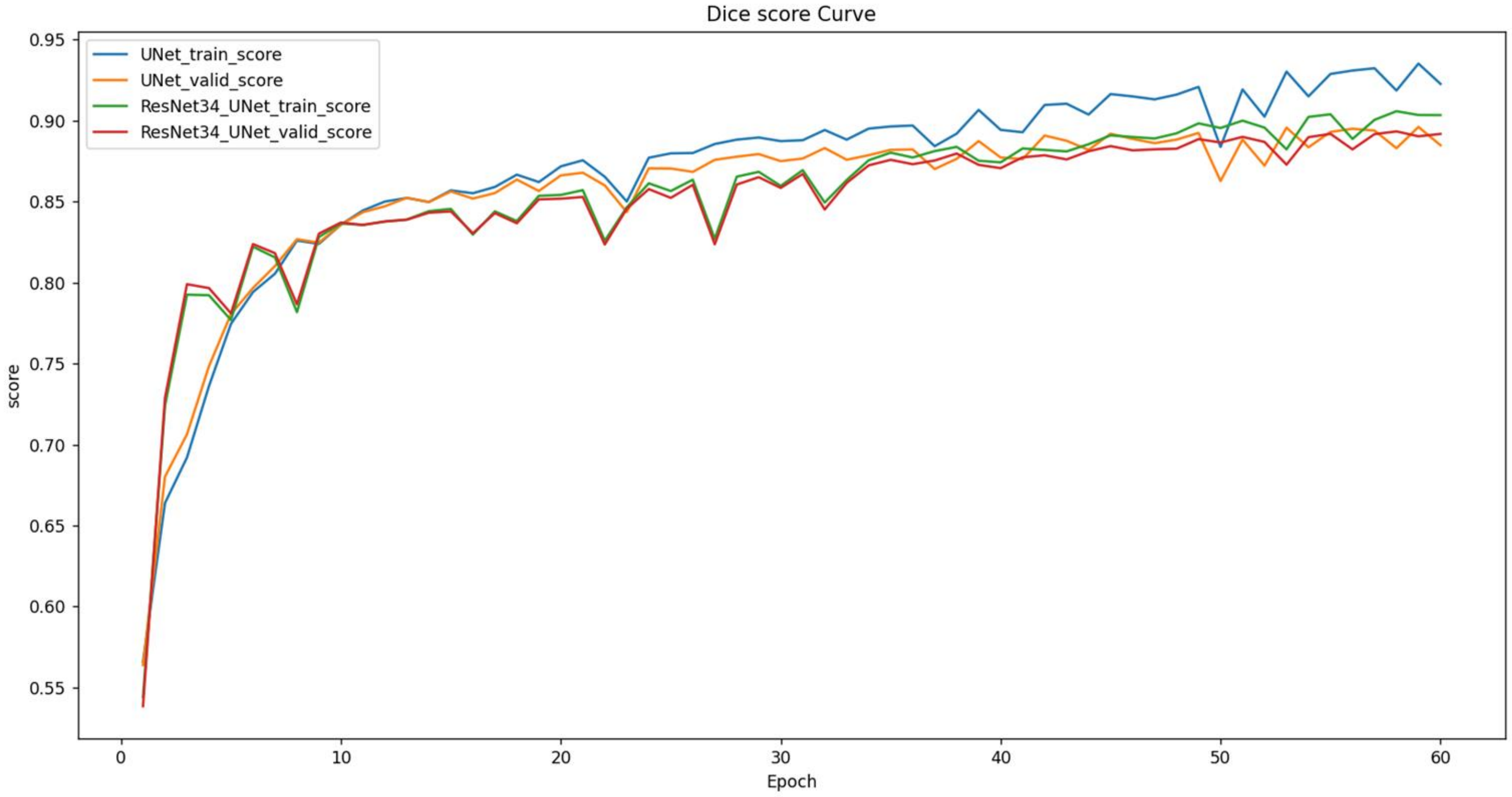
```
sample["image"] = (sample["image"]/255.0).astype(float)
mean = [0.485, 0.456, 0.406]
std = [0.229, 0.224, 0.225]
sample["image"][0] = ((sample["image"][0]-mean[0])/std[0]).astype(float)
sample["image"][1] = ((sample["image"][1]-mean[1])/std[1]).astype(float)
sample["image"][2] = ((sample["image"][2]-mean[2])/std[2]).astype(float)

return sample
```

What makes your method unique

由於我擔心在OxfordPetDataset裡用torch.transform會改變到mask，又不確定到底能將原本的程式碼改動到多少程度，所以我最後是在SimpleOxfordPetDataset裡，只用處理np.array的單純運算來處理資料。

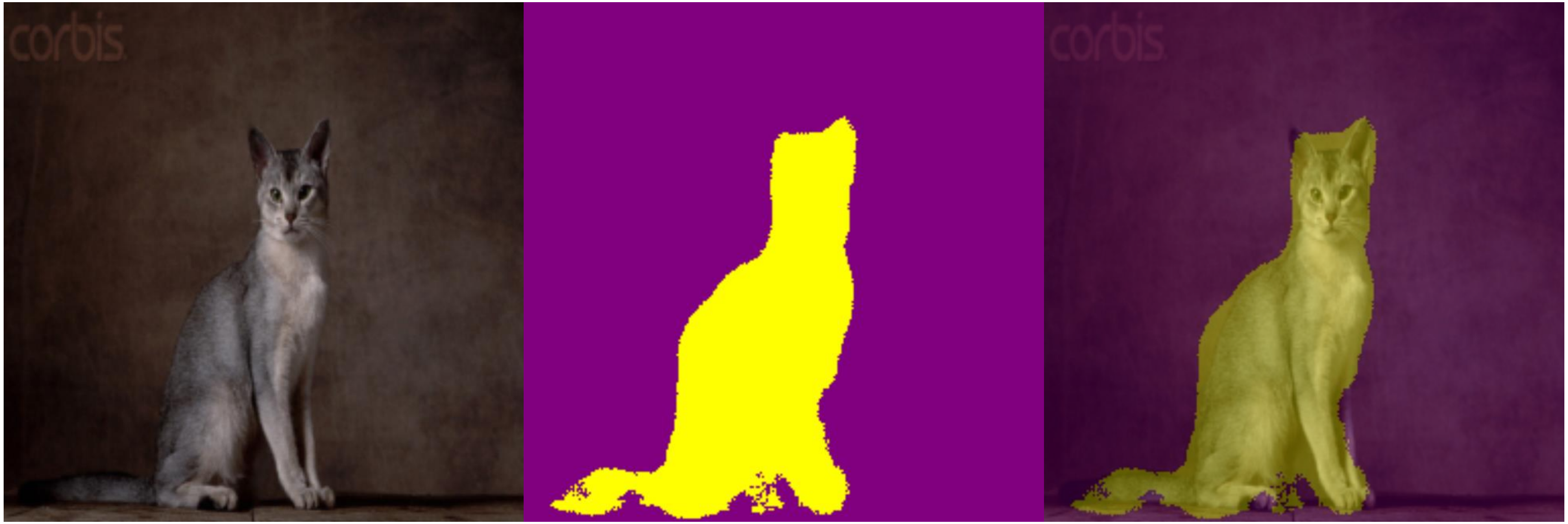
4. Analyze on the experiment results

Experiment results




```
----------UNet----------
UNet                | Train score:    0.92 | Valid score:    0.88 | Test score:    0.88
----------ResNet34_UNet----------
ResNet34_UNet       | Train score:    0.90 | Valid score:    0.89 | Test score:    0.89
```

Visualize the segmentation masks



What did you explore during the training process?

1. 我發現在epoch小於10時，valid的分數會大於train的分數，在epoch大於10後，train的分數才大於valid的分數。
推測是因為epoch小於10時，模型主要是學到圖片中間幾乎都是前景，對於前景背景的交界部分還沒有準確的預測，因此valid的分數剛好比較高。
epoch大於10後，模型對於前景背景的交界部分有概念了，因此對已經看過的train的資料集比較熟，對於沒看過的valid資料集就預測的分數比較差。
2. 在epoch小於10時，ResNet34_UNet的分數較高。

Found any characteristics of the data?

我發現在圖片中，前景幾乎都是在圖片中間，而圖片四周通常是背景，這會讓模型學習到：不管圖片中間的像素顏色如何，都很有可能是前景，而讓模型會需要像素顏色來判斷的地方，是前景與背景的交界部分。

5. Execution command

The command and parameters for the training process

--data_path：應該輸入放著dataset、saved_models、src的目錄路徑。

--model_name：應該輸入UNet或ResNet34_UNet，需注意大小寫。

--epochs、--batch_size、--learning_rate

例子：python train.py --model_name UNet

5. Execution command

The command and parameters for the inference process

--model：應該輸入存放在saved_models裡的模型名稱，例如：

UNet.pth或ResNet34_UNet.pth或DL_Lab3_UNet_312551133_鄧長

軒.pth，需注意大小寫。

--data_path ：應該輸入放著dataset、saved_models、src的目錄路徑。

例子：python inference.py --model UNet.pth

6. Discussion

What architecture may bring better results?

UNet與ResNet34_UNet的差別，在於當ResNet34作為encoder時，同時會有residual結構。

而我認為ResNet34_UNet會有比較好的預測結果，因為ResNet可透過residual來解決梯度消失爆炸的問題，使模型學習較為順利。

What are the potential research topics in this task?

1. 使用不同encoder與decoder的方法
2. 調整conv的參數
3. encode的次數、decode的次數
4. concatenation的方法
5. 尺寸縮放的倍率
6. channels的改變