台師大 613k0007c 余品誼

The main objective of this assignment is **binary semantic segmentation**, which involves classifying each pixel in an image as either foreground (the object of interest) or background. In this experiment, you will work with the **Oxford-IIIT Pet Dataset**, which contains images of cats and dogs, each accompanied by pixel-level annotations that precisely highlight the pet regions.

I. Implementation Details:

In this project, I implemented the training, evaluation, and inference code from scratch. The training process involves loading the dataset, applying data augmentation, and feeding the data into the model. A custom training loop was used, optimized with a combined loss function—**Binary Cross-Entropy (BCE) + Dice Loss**—to improve segmentation performance.

For model training, I experimented with two architectures: **UNet** and **ResNet34-UNet**. After training, the best-performing model was used to perform inference on a completely unseen image to validate its generalization capability.

The **Adam optimizer** was used during training, along with a **learning rate scheduler** to dynamically adjust the learning rate. The best model was selected and saved based on the highest **Dice Score** on the validation set.

```
# 定義損失函數
critertion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=args.learning_rate)
```

1. Model Architecture:

A. UNet Architecture:

The UNet architecture consists of a contracting encoder path and an expansive decoder path, each composed of four layers. The encoder captures feature maps at multiple resolutions, while the decoder progressively upsamples these features back to the original input size. Skip connections are used to concatenate feature maps from the encoder to the corresponding decoder layers, helping preserve spatial information.

Each basic building block of UNet includes two 3×3 convolutional layers, followed by ReLU activation functions to introduce non-linearity and enhance feature representation.

```
self.conv = nn.Sequential(
    #3x3捲積
    nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
    #加速收斂並穩定訓練
    nn.BatchNorm2d(out_channels),
    #加入非線性特性
    nn.ReLU(inplace=True),
    nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU(inplace=True)
)
```

```
#Encoder
#進行下採樣，保留編碼特徵
enc1 = self.enc1(x)
enc2 = self.enc2(self.pool(enc1))
enc3 = self.enc3(self.pool(enc2))
enc4 = self.enc4(self.pool(enc3))

#Bottleneck
bottleneck = self.bottleneck(self.pool(enc4))

#Decoder
#ConvTranspose2d 上採樣
dec4 = self.up4(bottleneck)
#將 Encoder 層的輸出與 Decoder 層的輸入拼接。
dec4 = torch.cat((enc4, dec4), dim=1)
dec4 = self.dec4(dec4)

dec3 = self.up3(dec4)
dec3 = torch.cat((enc3, dec3), dim=1)
dec3 = self.dec3(dec3)

dec2 = self.up2(dec3)
dec2 = torch.cat((enc2, dec2), dim=1)
dec2 = self.dec2(dec2)

dec1 = self.up1(dec2)
dec1 = torch.cat((enc1, dec1), dim=1)
dec1 = self.dec1(dec1)
```

B. ResNet34-UNet Architecture:

The **ResNet34-UNet** architecture utilizes **ResNet34** as the backbone encoder. ResNet34 is a 34-layer deep residual network commonly used for image recognition tasks. In this design, the layers of ResNet34 serve as the encoder, and **skip connections** are integrated into the decoder to retain spatial information.

The decoder uses **transposed convolution layers** to upsample the feature maps. At each level, features from the corresponding encoder

layers are concatenated with the decoder features to construct the full UNet architecture. To enable these skip connections between the encoder and decoder of two different models, **torch.nn.functional** operations are used to align the feature map dimensions before concatenation.

The basic UNet blocks remain largely the same as the original version, except that the first decoder layer is implemented using nn.ConvTranspose2d with a stride of 2 to perform upsampling.

```
dec5 = self.dec5(enc5)
dec5 = F.interpolate(dec5, size=enc4.shape[2:], mode="bilinear", align_corners=False)

dec4 = self.dec4(torch.cat([dec5, enc4], dim=1))
dec4 = F.interpolate(dec4, size=enc3.shape[2:], mode="bilinear", align_corners=False)

dec3 = self.dec3(torch.cat([dec4, enc3], dim=1))
dec3 = F.interpolate(dec3, size=enc2.shape[2:], mode="bilinear", align_corners=False)

dec2 = self.dec2(torch.cat([dec3, enc2], dim=1))
dec2 = F.interpolate(dec2, size=enc1.shape[2:], mode="bilinear", align_corners=False)

dec1 = self.dec1(dec2)
```

C. Training Setup and Optimization Strategy

The models were trained with a **batch size of 4**. Specifically, the UNet model was trained for **200 epochs**, while the ResNet34-UNet model was trained for **50 epochs**.

To address class imbalance in the dataset, a **combined loss function** was used—**Binary Cross-Entropy (BCE) Loss + Dice Loss**—which helps the model focus on both pixel-level accuracy and overlap between predicted and ground truth masks.

A **learning rate scheduler** was applied to dynamically adjust the learning rate, gradually reducing it in the later stages of training to help the model converge more effectively.

II. Data Preprocessing

In this section, we designed a custom data preprocessing pipeline using the **Albumentations** library for data augmentation. The goal is to increase data diversity and improve the model's generalization ability. The following

transformations were applied:

1. **Horizontal Flip**: Randomly flips the image horizontally with a probability of **50%**, helping the model learn symmetry in objects.
2. **Vertical Flip**: Randomly flips the image vertically with a probability of **20%**, enhancing the model's ability to handle vertically flipped inputs.
3. **Random Rotate 90**: Randomly rotates the image by 90 degrees with a probability of **30%**, allowing the model to learn directional invariance.
4. **ShiftScaleRotate**: A composite transformation that randomly shifts, scales, and rotates the image.
    A. shift_limit=0.1
    B. scale_limit=0.2
    C. rotate_limit=15
    D. p=0.5
    This transformation adds significant variety to the data in terms of position, size, and orientation.
5. **Random Brightness and Contrast Adjustment**: Randomly adjusts the image brightness and contrast with a probability of **20%**, making the model more robust to varying lighting conditions.
6. **Gaussian Blur**: Applies random Gaussian blur with a blur limit of **3** and a probability of **30%**, simulating images captured at different focus levels and sharpness.
7. **Normalization**: Standardizes the image by setting the pixel mean to **[0.5, 0.5, 0.5]** and standard deviation to **[0.5, 0.5, 0.5]**. This accelerates the training process and improves convergence.
8. **Conversion to PyTorch Tensor**: Finally, all transformed images are converted to **PyTorch tensors**, making them compatible with model training.

```python
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.VerticalFlip(p=0.2),
    A.RandomRotate90(p=0.3),
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=15, p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.GaussianBlur(blur_limit=3, p=0.3),
    A.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    ToTensorV2()  # 轉換為 PyTorch Tensor
])
```

III. Experimental Results and Analysis

1. train.py

A. ResNet34-UNet Architecture (200 Epochs)

Training Phase:

The blue line in the graph represents the Dice Score on the training set. As shown, the Dice Score increases rapidly with the number of training epochs and stabilizes at a high value, eventually approaching 1.0.

This consistent increase indicates that the model is learning the training data effectively.

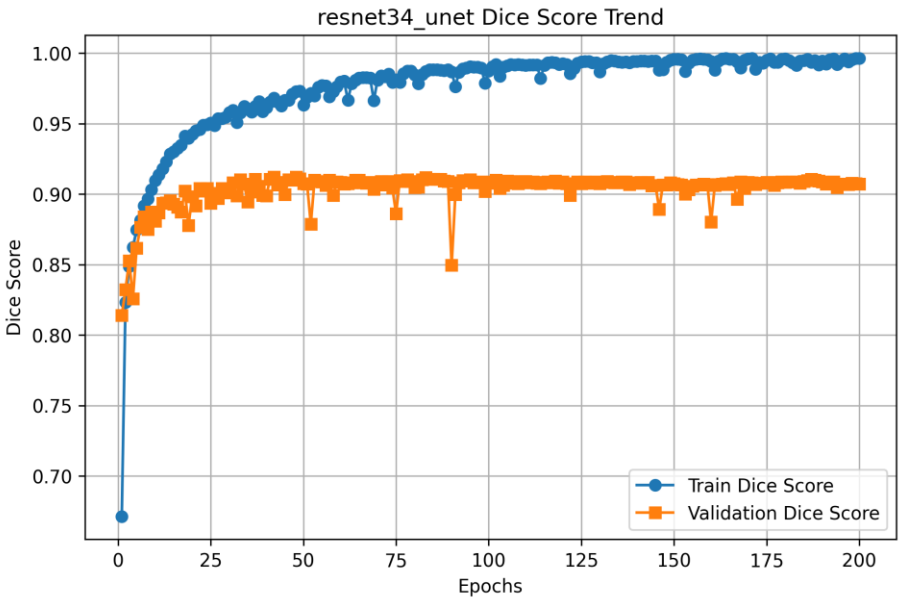
Validation Phase:

The orange line represents the Dice Score on the validation set.

Compared to the training set, the validation Dice Score fluctuates within a high range but does not show a steady upward trend.

Specifically, the validation Dice Score oscillates between 0.85 and 0.90, suggesting that the model's performance on unseen data has plateaued.

This could be an indication of overfitting, where the model performs very well on the training data but struggles to generalize to the validation data.



B. **UNet Architecture (200 Epochs)**

Training Phase:

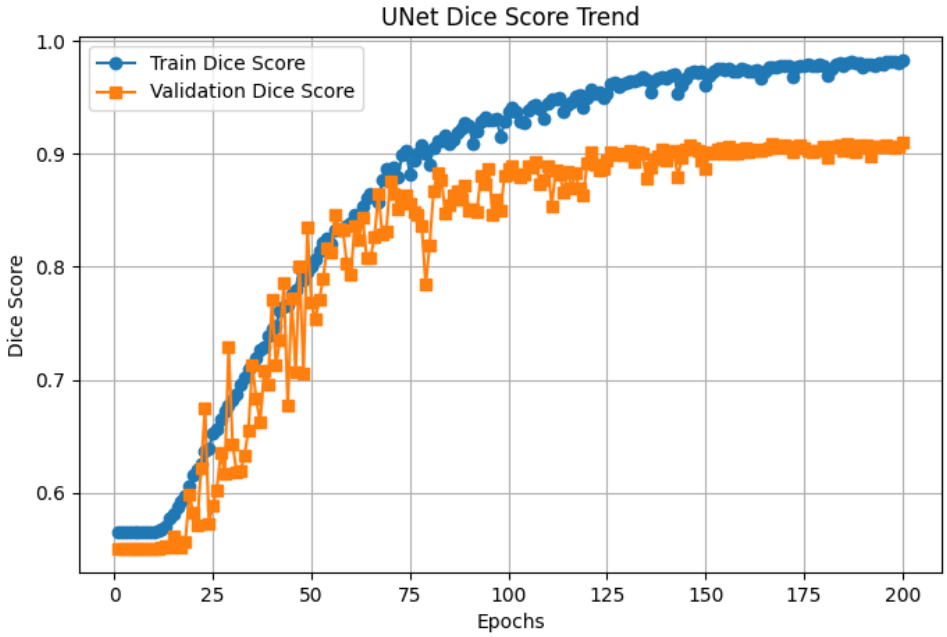
Throughout the 200 training epochs, the **Dice Score on the training set** steadily increased and approached **1.0**, indicating strong learning capability on the training data. This suggests that the model was able to effectively fit the training set.

Validation Phase:

The **Dice Score on the validation set** also showed an overall upward trend, although with more fluctuations compared to the training curve. After around **40 to 50 epochs**, the validation performance entered a relatively stable phase, eventually reaching around **0.9**.

This indicates that the model achieved a consistent level of performance on the validation set, although it did

not reach the same high accuracy as on the training set.



C. **Summary:**

Although **ResNet34-UNet** demonstrated more stable performance on both the training and validation sets, some signs of **overfitting** were still observed on the validation curve (as seen in minor fluctuations). This could be attributed to the model's increased complexity, making it more prone to fitting the training data too closely.

In contrast, the **UNet** model exhibited greater variability on the validation set, suggesting that **stronger regularization techniques** might be needed to improve its stability and generalization.

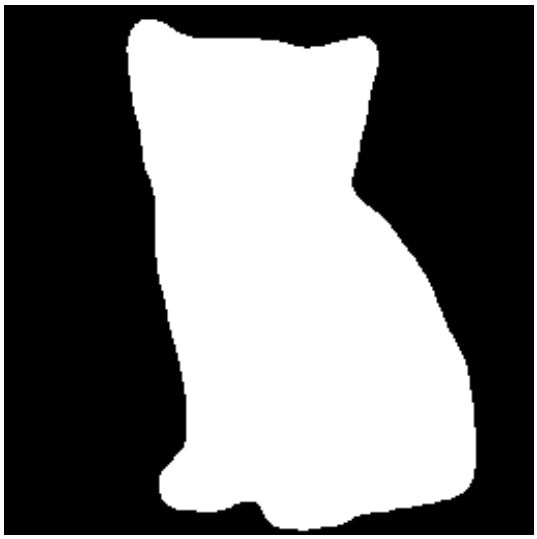
In conclusion, **ResNet34-UNet** outperformed UNet in terms of **stability and generalization**, while UNet may require further tuning to enhance its validation performance.

1. Inference.py

   kitten-1.jpg

A. resnet34_unet experimental results：

resnet34_unet_200_best_model.pth



Result Analysis:

The predicted contour of the cat is clear and well-defined, indicating that the model is capable of effectively identifying and segmenting the object within the image. This demonstrates a successful outcome for a binary semantic segmentation task, where the goal is to accurately distinguish the object of interest from the background.

B. Unet Experimental results：

best_model1-200.pth

Result Analysis:

Overall, the model successfully segmented the target object from the background. However, some **noise or unwanted artifacts** are still scattered around the outer regions of the prediction. This indicates that the **segmentation accuracy in certain areas** could be further improved, especially in refining object boundaries and reducing false positives.

IV. Execution Steps

1. Navigate to the src Directory

Before executing any scripts, make sure to navigate into the src directory:

cd src

2. Training Command and Parameters

Use the following command to train the model. You can specify the model type, dataset path, number of epochs, batch size, learning rate, and whether to download the dataset:

python train.py --model unet --data_path ../dataset/oxford-iiit-pet --epochs 200 --batch_size 4 --learning_rate 0.0001 --download no

3. Inference Command

To perform inference using a pre-trained model, use the following command. You can specify the model type, path to the saved model, input image, and output image path:

python inference.py --model resnet34_unet --model_path

"../saved_models/best_model1-50res34.pth" --data_path "./kitten-1.jpg" --output_path "./output.png"

V. Discussion

1. Model Performance Comparison

Based on the **Dice Score** observed during training, the **ResNet34-UNet** model achieved high training accuracy with **fewer epochs**, and both its training and validation results remained stable. This suggests that ResNet34-UNet possesses stronger feature learning capabilities, allowing it to learn faster and more effectively identify relevant patterns.

In contrast, the **UNet** model showed greater fluctuations during training, likely due to its relatively basic architecture lacking deep feature extraction layers like those in ResNet34. UNet exhibited **early instability** in training and **more volatile Dice Scores** on the validation set, indicating less consistent performance.

2. Inference Results

In the inference phase, both models were able to perform object segmentation to a reasonable degree. However, **ResNet34-UNet** generally produced **sharper and more accurate object boundaries**, capturing fine details more effectively.

While **UNet** also achieved decent segmentation, it tended to produce **blurred edges** in some finer regions—especially in complex or detail-rich images—where it sometimes misclassified parts of the object as background.

3. Future Optimization Strategies

A. Incorporate **deeper network architectures** such as ResNet or DenseNet to enhance the model's feature learning capacity.

B. Introduce additional **regularization techniques** (e.g., Dropout, advanced data augmentation) to reduce overfitting.

C. **Tune learning rates** or adopt more advanced optimizers such as **AdamW** to improve convergence and solution quality.