1. Implementation Details

a. UNet

The architecture used in this project is designed based on the framework Ronneberger, O., et al. [1]However, to ensure the following code can also support another UNet architecture using ResNet34 as the encoder and to avoid additional cropping when concatenating the encoder's feature maps during the up-convolution ,the convolution layers in the UNet are configured with stride=1, kernel_size=3, and padding=1 to maintain the feature map size unchanged. Additionally, a Dropout layer is added after the second convolution in each layer and after the up-convolution to prevent overfitting. The detailed modified architecture is shown in Figure 1. ◦

Each Double Convolution is described as following (Equation 1). The input x passes through Double Convolution layers, which consist of two consecutive 3x3 convolutions with Batch Normalization(BN) and ReLU activation funciton. F_{out} is the output feature map.

$F_{out} = ReLU(BN(Conv(x)))$	(1)

After each convolution block, the feature map undergoes 2x2 max pooling to halve the spatial resolution.

The whole encoder comprises four such stages, progressively encoding the feature representations from 64→128→256→512→1024 channels.

The decoder mirrors the encoder's structure to recover spatial information. Each upsampling stage employs ConvTranspose2d with a stride of 2, doubling the feature map's resolution. Next I concatenate feature maps from the corresponding encoder stage to the decoder. This operation helps recover fine-grained spatial details lost during max pooling.

The final layer is a 3x3 convolution that outputs a feature map with n_classes channels, maintaining the same spatial dimensions as the input. This allows pixel-wise classification.

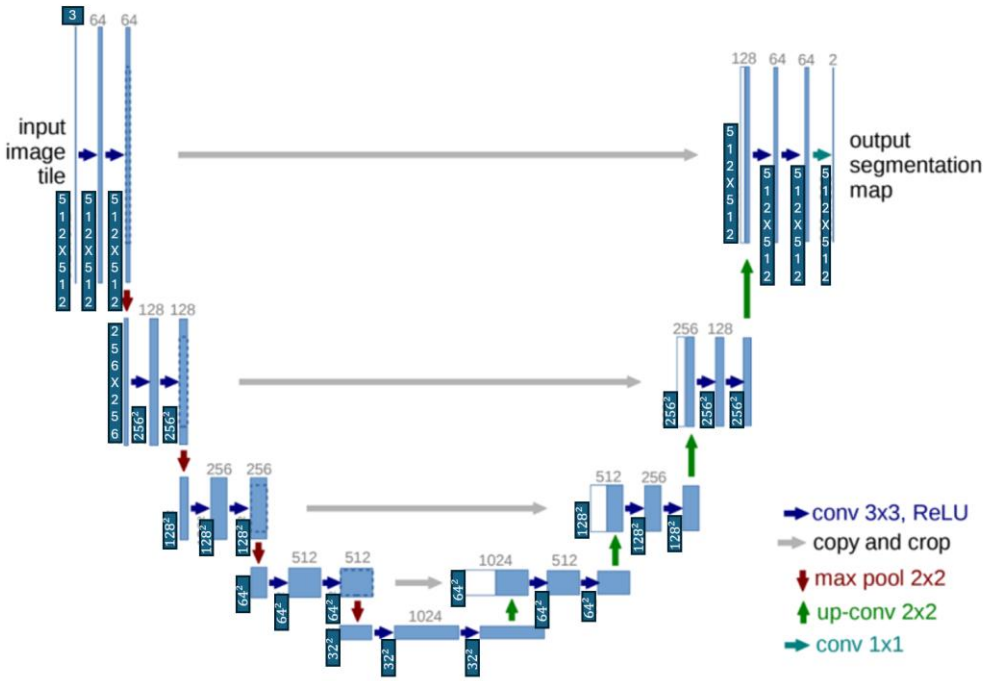


Figure 1. UNet framework

b. ResNet34_UNet

The ResNet34_UNet architecture is based on the framework proposed by Huang, Z., et al. [2]I omitted the pooling layers and added an extra encoder layer before concatenation to better match the feature dimensions. Additionally, a Dropout layer is added after the second convolution in each encoder block to prevent overfitting. The detailed architecture is shown in Figure 2.◦

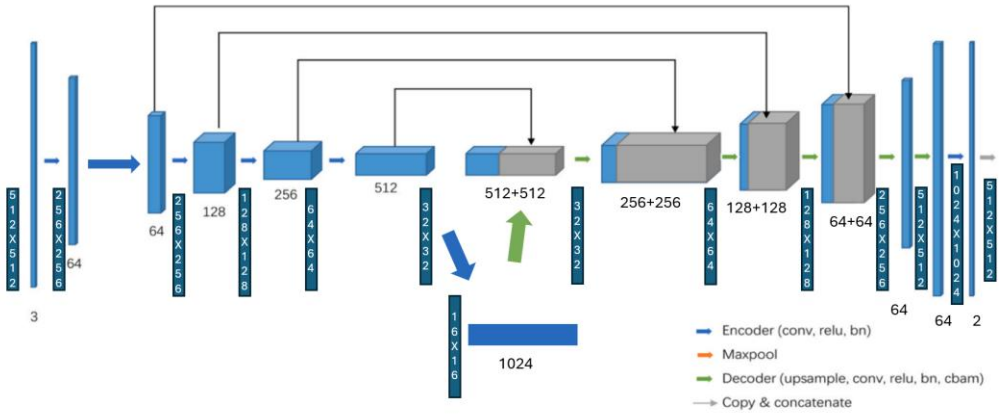


Figure 2. ResNet34UNet framework

(1) Encoder

The encoder part is inspired by the ResNet34 [3]architecture. By incorporating the design of residual blocks, it effectively mitigates the issues of vanishing gradients or exploding gradients. Each residual block consists of two 3x3 convolution layers, followed by Batch Normalization (BN) for regularization, and a final ReLU activation function, as shown in Equation (2). Specifically,$Conv2$ uses padding=1 and stride=1, while $Conv1$ has padding=1 and stride=1 only in the first encoder (which contains three residual blocks); for the remaining layers, it uses padding=1 and stride=2. A shortcut connection is introduced to maintain dimensional consistency, and if the input and output dimensions differ, a 1x1 convolution is used for adjustment.

$$F_{out} = ReLU(Conv2\left(ReLU\left(BN\left(Conv1(x)\right)\right)\right) + Shortcut(x)) \quad (2)$$

The detailed structure of each layer is as follows:

Initial convolution layer：

A 7×7 convolution layer processes the input image (3 channels), outputting a 64-channel feature map with stride=2 and padding=3 for initial feature extraction.

First encoder layer：

3 residual blocks, outputting a 64-channel feature map.

Second encoder layer：

4 residual blocks, reducing the feature map size and outputting 128 channels.

Third encoder layer：

6 residual blocks, outputting a 256-channel feature map.

Fourth encoder layer：

3 residual blocks, outputting a 512-channel feature map.

Fifth encoder layer：

3 residual blocks, outputting a final 1024-channel feature map.

The fifth layer is an additional layer designed to extend ResNet34 as the UNet encoder structure.

(2) Decoder

The decoder structure is almost identical to the original UNet decoder, so it won't be discussed in detail.

(3) Output Layer

The final output layer applies a 1x1 convolution to map the features to n_classes channels, ensuring the output size matches the original input. Additionally, F.interpolate is used to perform bilinear interpolation, resizing the output to 512×512 pixels.

c. Experiment Parameter Settings

In this project, the Dice score formula (Equation 2) is used as the evaluation metric, where N represents the number of images,y_{pred_i}denotes the predicted results , y_{true_i}represents the ground truth labels. For the loss function, cross entropy (Equation 3) is used. For an image of size $H \times W$, each pixel p has a ground truth label $y_{p,c} \in \{0,1\}$, and a predicted probability $\widehat{y_{p,c}} \in [0,1]$. Here, c represents the class, and in this project, there are two classes: foreground and background.

$$dice\ score = \frac{2\sum_{i=1}^{N} y_{pred_i} y_{true_i}}{\sum_{i=1}^{N} y_{pred_i} + \sum_{i=1}^{N} y_{true_i}} \quad (2)$$

$$\text{cross entropy} = -\frac{1}{H \times W} \sum_{p}^{H \times W} \sum_{c=1}^{C} y_{p,c} \log\left(\widehat{y_{p,c}}\right) \quad (3)$$

The optimizer supports three options: SGD, Adam, and AdamW, with weight_decay set to 1e-4 to prevent overfitting. The final result I choose Adam as my optimizer.

The learning rate is set to 0.001, and the MultiStepLR strategy is used to adjust the learning rate at [30, 50, 100, 150, 200] epochs, with a decay factor (gamma) of 0.1. This setup allows the model to converge quickly in the early stages of training while making finer weight adjustments later on.

Batch size is set to 8, and 4 worker threads (num_workers=4) are used to accelerate dataset loading, with prefetch_factor=2 to preload data. The total number of epochs is set to 50.

d. Training Process

In each epoch, the following steps are performed:

(1) Forward Propagation:

The input images are fed into the model to obtain predictions.

(2) Loss Calculation:

The outputs from step (1) are compared with the ground truth labels, and the loss is computed based on the chosen loss function.

(3) Backward Propagation:

The loss is backpropagated through the network to update gradients for each layer.

(4) Parameter Update:

The optimizer updates the model's weights based on the computed gradients.

(5) Learning rate Adjustment:

At the end of each epoch, the learning rate is adjusted according to the predefined schedule.

2. Data Preprocessing

Each image will be resized to 512X512 before training.

a. Data Augmentation

Before training the model, I applied several data augmentation techniques to the training dataset, including **random horizontal flip, random rotation, random scaling and translation, Gaussian blur, color shifting, and contrast adjustment**. All datasets including training, validation and tests datasets were **normalized.** These transformations help improve the model's performance and prevent overfitting. The implementation is based on the library developed by Buslaev, A.[4]

b. Image Preprocessing

Additionally, I performed preprocessing on the dataset and compared the results with those obtained without preprocessing before training. The preprocessing steps are as follows:

(1) Z-score Normalization

For each image in the dataset, I applied Z-score normalization to the RGB channels according to Equation (4)，Where X represents the pixel values of the image.，μ is the mean of the pixel values in the image.，σ is the standard deviation of the pixel values in the image.，ε is a small constant to prevent division by zero errors.

$$X' = \frac{X - \mu}{\sigma + \varepsilon}$$	(4)

(2) Min-max normalization

Next, to map the Z-score results back to values between 0 and 255, I applied Min-max normalization and multiplied by 255, as shown in Equation (5). Here, $\min(X')$represents the minimum pixel value in the image X', and $\max(X')$ represents the maximum pixel value in the imageX'。By combining Z-score normalization with Min-max normalization, this process helps eliminate differences between images caused by varying channel value ranges. It also enhances contrast by stretching the pixel values, correcting uneven contrast resulting from channel range differences.。

$$X_{norm} = \frac{X' - \min(X')}{\max(X') - \min(X')} \times 255$$	(5)

(3) Noise Removal

Next, I applied bilateral filtering, as shown in Equation (5), to the normalized images for noise removal. Compared to traditional Gaussian blur, this method smooths the internal textures while preserving the image edges, making it more suitable for the segmentation task in this project. In this process, p represents the target pixel. ，q represents a neighboring pixel around the target pixel. Ip is the color value of the target pixel. Iq is the color value of a neighboring pixel around the target pixel. S denotes the group of neighboring pixels around the target pixel. Gs is a Gaussian filter based on spatial distance (weights pixels according to their distance from the target pixel). Gr is a Gaussian filter based on color difference (weights pixels according to their color similarity to the target pixel).$1/W_p$ is a normalization factor, which is set to 5 in this project.

$$I_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(\|I_p - I_q\|) I_q$$	(6)

(4) Local Brightness Enhancement

Finally, to further enhance details in the darker regions of the image, I applied the CLAHE (Contrast Limited Adaptive Histogram Equalization) technique. Unlike traditional histogram equalization, CLAHE divides the image into multiple small regions and performs contrast equalization independently within each region. It also sets an upper limit on contrast to prevent noise amplification caused by over-enhancement. In this process, The image is divided into an 8×8 grid of regions. A clipping limit of 2.0 is set to ensure that dark details are enhanced while preventing brighter areas from becoming overexposed.

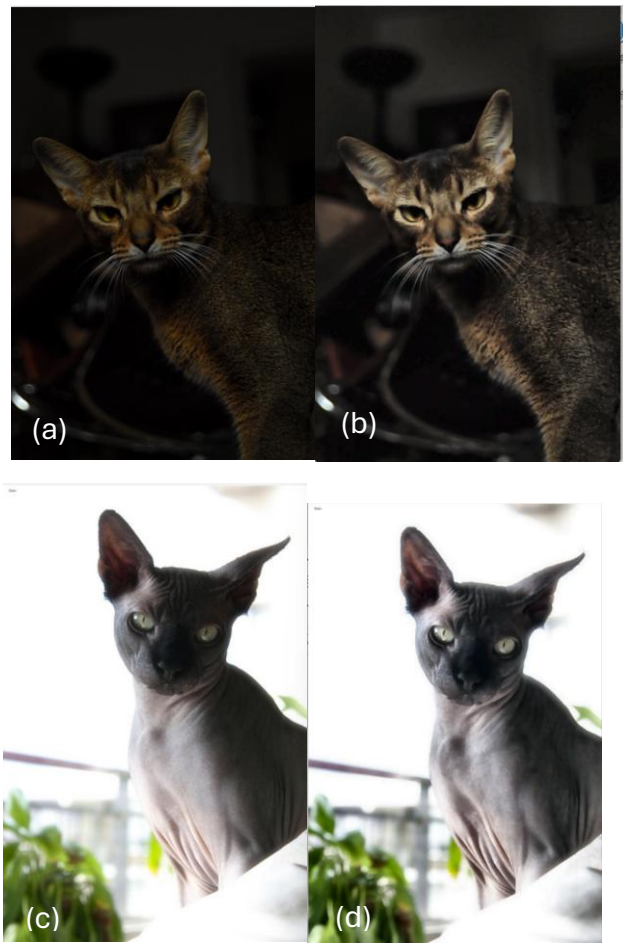


Figure 3 (a,c) original images (b,d) images after preprocessing

From Figure 3, it can be seen that the brightness and darkness of the image are more balanced after preprocessing, and the contrast is more pronounced.

All implementations were carried out using OpenCV.[5] However, during the experiment, some non-image files and corrupted images that could not be read were identified, including:

Abyssinian_102.mat

Egyptian_Mau_167.jpg

Egyptian_Mau_177.jpg

Abyssinian_101.mat

Egyptian_Mau_145.jpg

Egyptian_Mau_139.jpg

Abyssinian_34.jpg

Abyssinian_100.mat

Egyptian_Mau_191.jpg

Therefore, during the implementation, the program will print the filenames of these files and skip them.

3. Analyze the experiment results

From Table 1, it can be observed that the model with preprocessing performed better than the model without preprocessing. Although the difference is small (0.01-0.04), better results could potentially be achieved by increasing the number of training epochs in the future.

When comparing the ResNet34UNet architecture with the UNet architecture, it's clear that ResNet34UNet outperforms UNet in both cases — with and without preprocessing. I hypothesize that this is due to the Residual structure in ResNet, which helps the model retain low-dimensional features while extracting higher-dimensional ones, resulting in better performance compared to the original UNet architecture.

Models	Average Dice score/test
ResNet34UNet(No preprocessing)	0.9318733
ResNet34UNet(With preprocessing)	**0.93587244**
UNet(No preprocessing)	0.8886922
UNet(With preprocessing)	0.89453226

Table 1 Model Performance on the Test Set

From the perspective of model training, as shown in Table 2, both model architectures demonstrate more stable convergence after incorporating preprocessing compared to training without preprocessing. This effect is particularly evident in the UNet architecture, where the validation loss fluctuations become smoother after preprocessing, indicating the effectiveness of the preprocessing steps.

Furthermore, comparing the performance between the ResNet34UNet model and the UNet model, it is observed that ResNet34UNet consistently achieves faster convergence and lower final loss, regardless of whether preprocessing is applied. This demonstrates that using ResNet34 as the encoder provides stronger

feature extraction capabilities, contributing to better overall model performance.

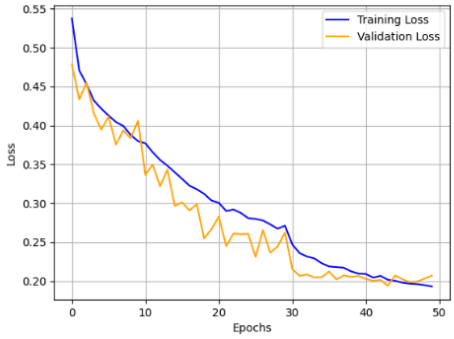
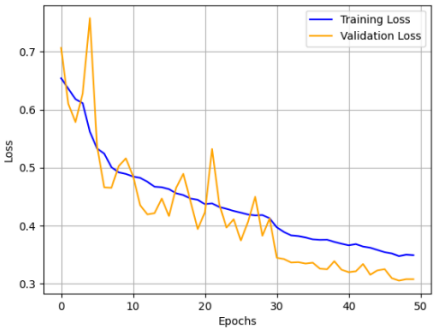
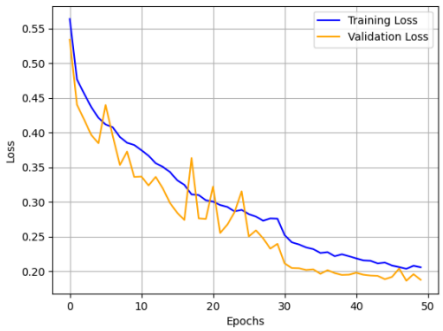
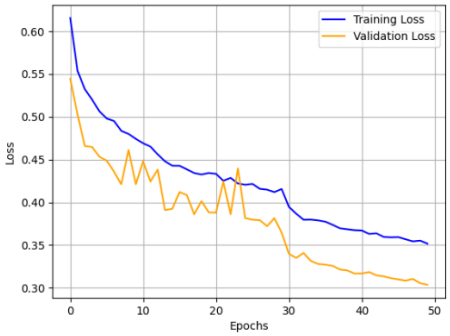
ResNet34UNet(No preprocessing)	UNet(No preprocessing)
	
ResNet34UNet(With preprocessing)	UNet(With preprocessing)
	

Table 2 Model Loss on the Train/Val Set

From the visual results of the test set, as shown in Table 3, it can be observed that the masks predicted by the ResNet34UNet architecture exhibit smoother edges and more accurate structural contours compared to the standard UNet architecture. This improvement is especially noticeable in details such as ears and tails, demonstrating the stronger feature extraction capability of ResNet34UNet.
The data with preprocessing shows an improvement in segmentation results for both architectures. The experimental results indicate that ResNet34UNet effectively suppresses background misclassification after preprocessing. While standard UNet also shows a slight improvement in background suppression, it remains less effective than ResNet34UNet, highlighting the need for further enhancement in learning deep features.

The bottom row of images illustrates scenarios with complex backgrounds. In these cases, UNet fails to accurately delineate the object's contours, regardless of

whether preprocessing is applied. However, although ResNet34UNet still makes some local errors, it manages to produce a more complete and recognizable outline, further showcasing its effectiveness.

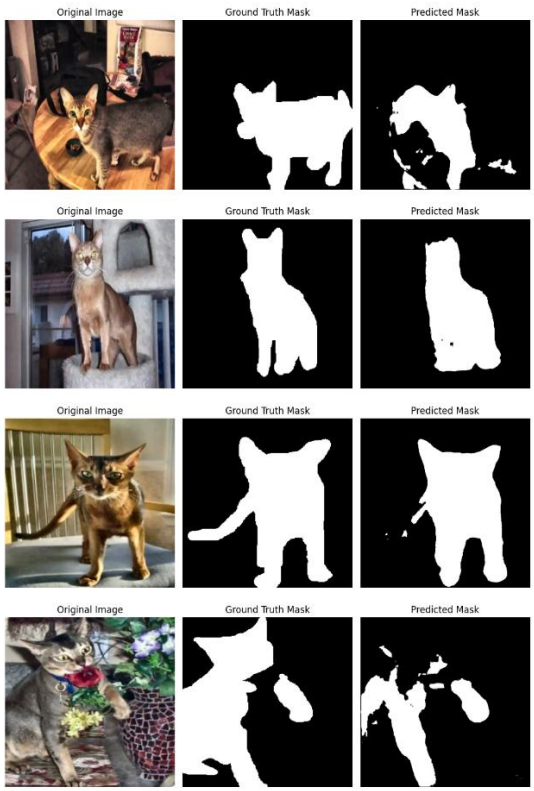
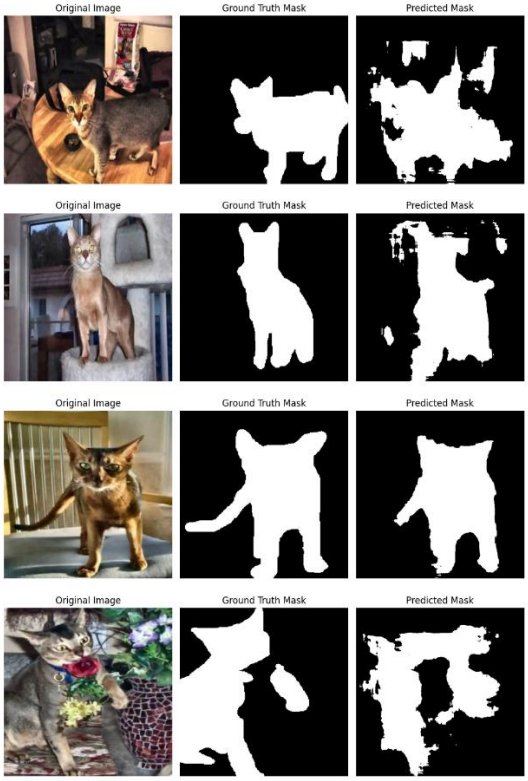
ResNet34UNet(No preprocessing)	UNet(No preprocessing)
	
ResNet34UNet(With preprocessing)	UNet(With preprocessing)

Table 3 Examples of Segmentation in test set

4. Execution steps

(1) Unzip DL_Lab2_s1103344_陳姿蓉.zip

(2) Navigate to DL_Lab2_s1103344_陳姿蓉/

(3) Open cmd and type python exe.py

(4) After running exe.py, the script will automatically download the dataset, perform preprocessing, and train four models: UNet (without preprocessing), UNet (with preprocessing), ResNet34UNet (without preprocessing), and ResNet34UNet (with preprocessing). It will then test all four models. Parameters such as epochs can be adjusted directly in exe.py.。

5. Discussion

Based on the experimental results, it can be observed that using ResNet34 as the encoder achieves better performance compared to the standard UNet. When combined with appropriate preprocessing, the model's performance improves significantly. This demonstrates that changing the encoder can lead to varying outcomes for the model.， Xu, G., et al.[6]achieved impressive results in medical image segmentation by using a Transformer as the encoder for UNet. Therefore, it

might be worth exploring Transformers as encoders in future work. Additionally, the preprocessing methods used in this experiment can be further enhanced. For instance, converting images to grayscale for training might help reduce misclassification caused by similar colors in the background and foreground. Moreover, techniques like Canny or other edge detection methods could be employed to emphasize edge textures and improve segmentation performance.

References

[1] Ronneberger, O., P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. 2015. Cham: Springer International Publishing.

[2] Huang, Z., et al., *Deep learning-based pelvic levator hiatus segmentation from ultrasound images*. European Journal of Radiology Open, 2022. **9**: p. 100412.

[3] He, K., et al. *Deep Residual Learning for Image Recognition*. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[4] Buslaev, A., et al. *Albumentations: Fast and Flexible Image Augmentations*. Information, 2020. **11**, DOI: 10.3390/info11020125.

[5] Bradski, G. and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. 2008: " O'Reilly Media, Inc.".

[6] Xu, G., et al. *LeViT-UNet: Make Faster Encoders with Transformer for Medical Image Segmentation*. in *Pattern Recognition and Computer Vision*. 2024. Singapore: Springer Nature Singapore.