Deep Learning Lab3: Binary Semantic Segmentation

ID: 412551030   Name: 黃羿寧

1. Overview of the lab 3

Deep learning has been widely used in several fields such as image recognition, medical diagnosis and natural language processing. In field of binary semantic segmentation, UNet[1] was the most common used model for successfully distinguish our region of interest, such as object location. In recent years, a replacement of ResNet34 in the encoder of a Unet, namely UNet-ResNet34 has further improved the generalizability, capability and performance[2]. To deeper understand the architecture, training process and comparison between two aforementioned models, this lab requests to build the model and train them from scratch to successfully distinguish the cat & dogs location compared to the background. In this report, I will present the implementation details of two models, methods for data preprocessing, details of my code, and the results gained from the practice. In this practice, UNet and ResUnet showed a dice score of 88.1% and 88.21% on testing dataset, showing the powerful prediction of two models. Dataset: Oxford-IIIT Pet Dataset (https://www.robots.ox.ac.uk/~vgg/data/pets/)

2. Implementation Details

A. Details of your training, evaluating, inferencing code

(1) Training.py

Training consists of a pre-defined model, data, loss (criterion), optimizer and device in the main.py. The training process specifically includes a `torch.squeeze(labels, 1)` to squeeze the channel dimensions in order to correctly calculate the loss. And most importantly, I delete the `images, labels, outputs, loss` in order to save the memory during the training, since I did not have enough GPU memory at the first try.

```
def train(model, data, criterion, optimizer, device):
    # Start training
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for images_dist in data:
        images = images_dist['image'].to(device)
        labels = images_dist['mask'].to(device)
        optimizer.zero_grad()
        outputs = model(images)
        labels = torch.squeeze(labels, 1)
        outputs = torch.squeeze(outputs, 1)
        loss = criterion(outputs, labels)
        # Backward pass and optimize
        loss.backward()
        optimizer.step()
        del images, labels, outputs, loss
        torch.cuda.empty_cache()
```

(2) Evaluate.py

`evaluate(model, data, criterion, device)`particularly being used in main.py. The process contains calculation of loss and dice score. The dice score was calculated using the formula provided by the lab requirements, following the formula of:

$$Dice\ score\ = (\frac{2\ *\ (number\ of\ common\ pixels)}{(predicted\ img\ size\ +\ groud\ truth\ img\ size)})$$

Hence, I first calculate the intersection of prediction and GT and divide the number of common pixels by the total image size to get the dice score of the prediction.

```python
def dice_score(output, label):
    output = torch.sigmoid(output)
    preds = output > 0.5
    label = label > 0.5
    intersection = (preds & label).float().sum()
    total_size = preds.float().sum() + label.float().sum()

    dice = (2. * intersection+1e-6) / (total_size+1e-6)
    return dice.mean()

def evaluate(model, data, criterion, device):
    model.eval()
    running_loss = 0.0
    dice_scores = []
    for images_dist in data:
        # Move data to the appropriate device (CPU or GPU)
        images = images_dist['image'].to(device)
        labels = images_dist['mask'].to(device)
        # get updated model prediction
        outputs = model(images)
        labels = torch.squeeze(labels, 1)
        outputs = torch.squeeze(outputs, 1)
        # Calc updated loss
        loss = criterion(outputs, labels)

        running_loss += loss.item()
        dice_score_batch = dice_score(outputs, labels)
        dice_scores.append(dice_score_batch.item())

        del images, labels, outputs, loss, dice_score_batch
        torch.cuda.empty_cache()

    average_dice_score = sum(dice_scores) / len(dice_scores)
    return running_loss/len(data), average_dice_score
```

(3) Inference.py

The testing will be used by `test(TestData, Model1, hist_res, savepath_Res, device)` . The hist_res was a dictionary storing the training accuracy, training loss, validation accuracy, validation loss and so on. Therefore, I will use the `best_epoch_test(hist_res)` to find the best epoch of the highest validation accuracy. Then, I will load the model weight and use the evaluation function to test on testing datasets.

```python
def best_epoch_test(hist_res):
    val_acc_array = np.array(hist_res["val_acc"])
    best_epoch = val_acc_array.argmax()
    return best_epoch
```

```
def test(TestData, Model1, hist_res, savepath_Res, device):
    best_epoch1 = best_epoch_test(hist_res)
    test_model_path1 = os.path.join(savepath_Res, "{}-ep{}.pth".format("Model", best_epoch1))
    checkpoint = torch.load(test_model_path1, map_location="cpu")
    Model1.load_state_dict(checkpoint["state_dict"])
    loss_fn = nn.CrossEntropyLoss()
    # evaluate(model, data, criterion, device)
    _, testacc = evaluate(Model1, TestData, loss_fn, device)
    print("(argmax valid ep = {}): Train dice ={:.4f}, Valid dice ={:.4f}, Test
dice={:.4f}".format(best_epoch1, hist_res["acc"][best_epoch1],hist_res["val_acc"][best_epoch1],
testacc))
```

B. Details of your model (UNet & ResNet34_UNet)

(1) UNet.py

Due to the time and device constraints, UNet[1] applied in this study consists of **three times** of down sampling and up sampling. The down sampling was conducted by `MaxPool2d((2,2),stride = 2` and the up sampling was conducted by `ConvTranspose2d(512, 256, (2,2),stride = 2)` . Basically, the UNet stores and concatenates every variable after convolution to the corresponding one in the up-sample process.

```
class ConvTwice(nn.Module):
    def __init__(self, ch_in, ch_out):
        super().__init__()
        self.ConvTwice = nn.Sequential(
            nn.Conv2d(ch_in, ch_out, (3,3), padding=1),
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace = True),
            nn.Conv2d(ch_out, ch_out, (3,3), padding=1),
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace = True))

    def forward(self, x):
        return self.ConvTwice(x)

class Unet(nn.Module):
    def __init__(self):
        super(Unet, self).__init__()
        self.conv1_down = nn.Sequential(ConvTwice(3,64))
        self.conv2_down = nn.Sequential(nn.MaxPool2d((2,2),stride = 2),
            ConvTwice(64,128))
        self.conv3_down = nn.Sequential(nn.MaxPool2d((2,2),stride = 2),
            ConvTwice(128,256))
        self.conv4_down = nn.Sequential(nn.MaxPool2d((2,2),stride = 2),
            ConvTwice(256,512))
        self.conv1_up1 = nn.ConvTranspose2d(512, 256, (2,2),stride = 2)
        self.conv1_up2 = ConvTwice(512, 256)
        self.conv2_up1 = nn.ConvTranspose2d(256, 128, (2,2),stride = 2)
        self.conv2_up2 = ConvTwice(256, 128)
        self.conv3_up1 = nn.ConvTranspose2d(128, 64, (2,2),stride = 2)
        self.conv3_up2 = ConvTwice(128, 64)

        self.fullyconnect = nn.Conv2d(64, 1, (1,1))

    def forward(self, x):
        x1 = self.conv1_down(x)
        x2 = self.conv2_down(x1)
        x3 = self.conv3_down(x2)
        x4 = self.conv4_down(x3)

        x = self.conv1_up1(x4)
        x = self.conv1_up2(torch.cat([x, x3], dim=1))

        x = self.conv2_up1(x)
        x = self.conv2_up2(torch.cat([x, x2], dim=1))

        x = self.conv3_up1(x)
        x = self.conv3_up2(torch.cat([x, x1], dim=1))

        pred = self.fullyconnect(x)
        return pred
```

(2) Resnet34_Unet.py

The Resnet34_Unet follows the ResNet structure developed in lab2. The `conv_lay` function was typically used in the duplicated format in the bottleneck of the ResNet model. And the `ConvTwice` is also used in this model since the decoder remains identical to the UNet.

```
Class conv_lay(nn.Module):
    """Left block + shortcut template"""
    def __init__(self, Input_Channel, Output_Channel, stride = 1, downsample=None):
        super(conv_lay, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(Input_Channel, Output_Channel, (1,1), 1, bias=False),
            nn.BatchNorm2d(Output_Channel),
            nn.ReLU(inplace = True))
        self.conv2 = nn.Sequential(
            nn.Conv2d(Output_Channel, Output_Channel, (3,3), stride = stride, bias=False,
padding=1),
            nn.BatchNorm2d(Output_Channel))

        self.ReLu = nn.ReLU(inplace = True)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        if self.downsample is not None:
            identity = self.downsample(x)

        Output = self.conv1(x)
        Output = self.conv2(Output)

        # Apply shortcut
        Output += identity
        # Relu
        Output = self.ReLu(Output)
        del x, identity
        return Output

class ConvTwice(nn.Module):
    def __init__(self, ch_in, ch_out):
        super().__init__()
        self.ConvTwice = nn.Sequential(
            nn.Conv2d(ch_in, ch_out, (3,3), padding=1),
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace = True),
            nn.Conv2d(ch_out, ch_out, (3,3), padding=1),
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace = True))

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

```
class U_ResNet34(nn.Module):
    def __init__(self):
        super(U_ResNet34, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, (7,7), stride = 2, bias = False, padding = 3),  # [64, 128, 128]
            nn.BatchNorm2d(64),
            nn.ReLU(inplace = True),
            nn.MaxPool2d(kernel_size = (3,3), stride = 1, padding=1),      # [64, 64, 64]
        )
        self.input_channel_size = 64

        self.conv2_x = self.layer_iteration(3, 64, 1)
        self.conv3_x = self.layer_iteration(4, 128, 2)
        self.conv4_x = self.layer_iteration(6, 256, 2)
        self.conv5_x = self.layer_iteration(3, 512, 2)
        self.conv1_up1 = nn.ConvTranspose2d(512, 256, (2,2),stride = 2)
        self.conv1_up2 = ConvTwice(512, 256)
        self.conv2_up1 = nn.ConvTranspose2d(256, 128, (2,2),stride = 2)
        self.conv2_up2 = ConvTwice(256, 128)
        self.conv3_up1 = nn.ConvTranspose2d(128, 64, (2,2),stride = 2)
        self.conv3_up2 = ConvTwice(128, 64)

        self.fullyconnect = nn.Sequential(
            nn.ConvTranspose2d(64, 64, (2,2),stride = 2),
            nn.Conv2d(64, 1, (1,1)))

    def layer_iteration(self, session, nodenum, stride = 1):
        downsample = None
        if stride != 1 or self.input_channel_size != nodenum:
            downsample = nn.Sequential(
            nn.Conv2d(self.input_channel_size, nodenum, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(nodenum))

        ML = []
        ML.append(conv_lay(self.input_channel_size, nodenum, stride = stride,
downsample=downsample))
        self.input_channel_size = nodenum

        for i in range(1, session):
            ML.append(conv_lay(self.input_channel_size, nodenum))

        return nn.Sequential(*ML)

    def forward(self, x):
        x = self.conv1(x)
        x1 = self.conv2_x(x)
        x2 = self.conv3_x(x1)
        x3 = self.conv4_x(x2)
        x4 = self.conv5_x(x3)
        x = self.conv1_up1(x4)
        x = self.conv1_up2(torch.cat([x, x3], dim=1))
        x = self.conv2_up1(x)
        x = self.conv2_up2(torch.cat([x, x2], dim=1))
        x = self.conv3_up1(x)
        x = self.conv3_up2(torch.cat([x, x1], dim=1))
        pred = self.fullyconnect(x)
        del x, x1, x2, x3, x4
        return pred
```

C. Main execution code

The main.py code is used for training all the models, for details please see section 5. The device used in this study is NVIDIA GeForce RTX 4090. For the model training, the parameters used include a batch size of 16 (1^{st} try) & 32 (2^{nd} try), and the Adam optimizer with a learning rate of 0.001 (1^{st} try)

& 0.0001 (2nd try) and a weight decay of 0.0001. There will be a comparative analysis on the parameter used in the result part. The loss is calculated using cross-entropy. The training iterations were set at 150 in the first attempt and at 200 in the second attempt.

```
if __name__ == "__main__":

    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(device)
    # Enable cuDNN benchmarking
    torch.backends.cudnn.benchmark = True

    # model setting
    a = int(input('Which model r u going to train? (Unet = 0), (ResUnet = 1)'))
    if a == 0:
        Model1 = Unet().to(device)
    else:
        Model1 = U_ResNet34().to(device)
    # load data
    print('----Start Data load----')
    current_path =  os.getcwd()
    data_path = os.path.join(current_path,"dataset", "oxford-iiit-pet")
    batch_size = 32
    num_workers = 4
    TrainData = load_dataset(data_path, 'train', batch_size, num_workers)
    print('{} images are loaded for training'.format(len(TrainData)*batch_size))
    ValidData = load_dataset(data_path, "valid" , batch_size, num_workers)
    print('{} images are loaded for validation'.format(len(ValidData)*batch_size))
    TestData = load_dataset(data_path, "test" , 1, num_workers)
    print('{} images are loaded for testing'.format(len(TestData)))

    # training epochs
    num_epochs = 200
    # start training
    loss_fn = nn.CrossEntropyLoss()
    lr = 1*1e-4
    opt_fn = optim.Adam(Model1.parameters(), lr=lr, weight_decay=1e-4)

    hist_UNet = dict(
        loss=np.zeros((num_epochs, )), val_loss=np.zeros((num_epochs, )),
        acc=np.zeros((num_epochs, )), val_acc=np.zeros((num_epochs, )))

    print('-----start_training-----')
    if a == 0:
        savepath_unet = os.path.join('saved_models', 'Unet_0408')
        os.makedirs(savepath_unet, exist_ok=True)
    else:
        savepath_unet = os.path.join('saved_models', 'ResUnet_0408')
        os.makedirs(savepath_unet, exist_ok=True)

    for epoch in range(num_epochs):
        # train(model, data, criterion, optimizer, device):
        train(Model1, TrainData, loss_fn, opt_fn, device)
        # evaluate(model, data, criterion, device):
        loss, acc = evaluate(Model1, TrainData, loss_fn, device)
        val_loss, val_acc = evaluate(Model1, ValidData, loss_fn, device)
        print("Epoch {}: loss={:.4f}, acc={:.4f}, val_loss={:.4f}, val_acc={:.4f}".format(epoch,
loss, acc, val_loss, val_acc))
        hist_UNet["loss"][epoch] = loss
        hist_UNet["acc"][epoch] = acc
        hist_UNet["val_loss"][epoch] = val_loss
        hist_UNet["val_acc"][epoch] = val_acc
        if True:
            checkpoint = {'epoch': epoch,'state_dict': Model1.state_dict(),'optimizer':
opt_fn.state_dict(),'loss': loss, 'acc': acc, 'val_loss': val_loss, 'val_acc': val_acc}
            torch.save(checkpoint, os.path.join(savepath_unet, f"Model-ep{epoch}.pth"))
    np.savez(os.path.join(savepath_unet, 'training_process.npz'), **hist_UNet)
    print('-----File Saved-----')
```

3. Data Preprocessing

A. Steps of data preprocessing

Data preprocessing is crucial for the image data used in DL models. The primary goal of this preprocessing isn't just to improve image quality and classification accuracy. It's also to enhance the model's ability to generalize, ensuring it performs well even with complex datasets. Typical methods for preprocessing image data include resizing, rotation, color correction, contrast enhancement, and noise reduction, among others[3]. In this lab, the size of the data was resized to (256, 256, 3). Furthermore, to increase the classification capability of the mode, I applied **color jitter** to the train and the validation dataset in the model training. The transformation of color randomly applied jitter to **brightness, contrast and saturation** to the original figure. The example below showed the 4 figures applied function. However, since the datasets provided ground truth of the mask, I did not apply any rotation, flipping methods.

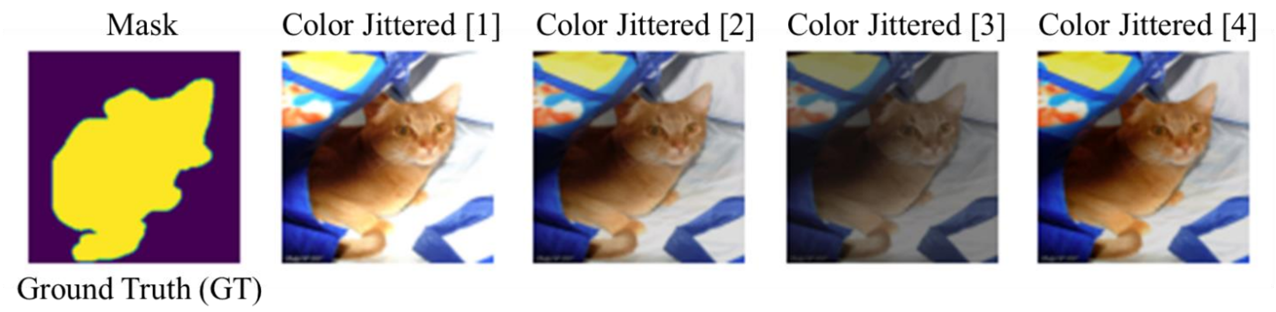


Figure 1. The example of color jittered figures

B. What make this method special?

The color jittering method was widely used in data augmentation, and a recent study has shown its capability to encourage the shape learning [4].Therefore, I used this method originally aiming to increase the amount of training data (data augmentation) and lead to the raises of model's generalizability and the performance. However, due to the time constraint, I only trained and validated on jittered once data.

4. Analyze on the experiment results

A. Comparison figures

In this lab, I trained the two model twice in different batch size, learning rates (lr) and amount of iterations. The first try uses a batch size of 16, lr of 0.001 and 150 iterations. As we can see in the Figure 2 (left), the result of dice score shows a little difference between training and validation in the fluctuated learning curve. And it is apparently not achieving the ceil. Interestingly, I observed that the UNet trains faster but start overfitting to training datasets around 20-30 epochs in this combination. This shows the importance of inclusion of early-stopping to reduce the computational cost. However, since the dice score of the testing dataset did not satisfy the lab requirement, I modified some parameters and trained again from scratch. In the second try, I changed the batch size to 32, lr to 0.0001 and expand the training epochs to 200. The Figure 2 (right) displays the learning curve of the two models in two datasets. The results showed smother trend and higher average dice score in the validation dataset. Therefore, I selected the combination of (batch = 32, lr = 0.0001) to predict the boundaries in the testing datasets.

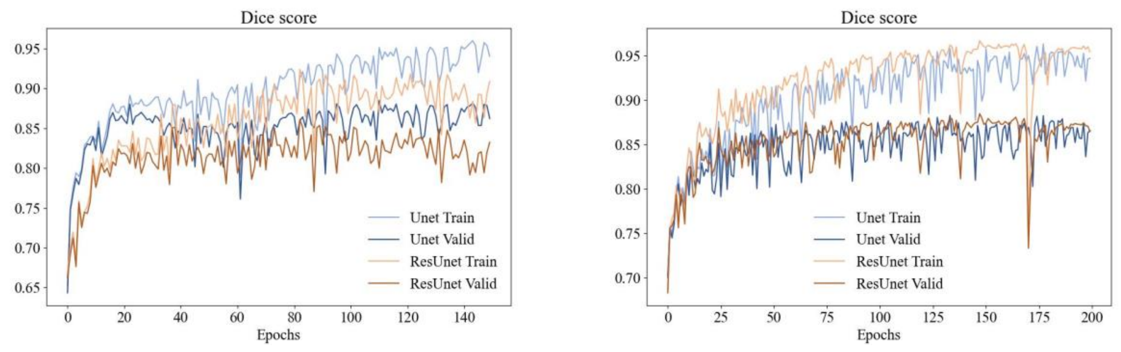
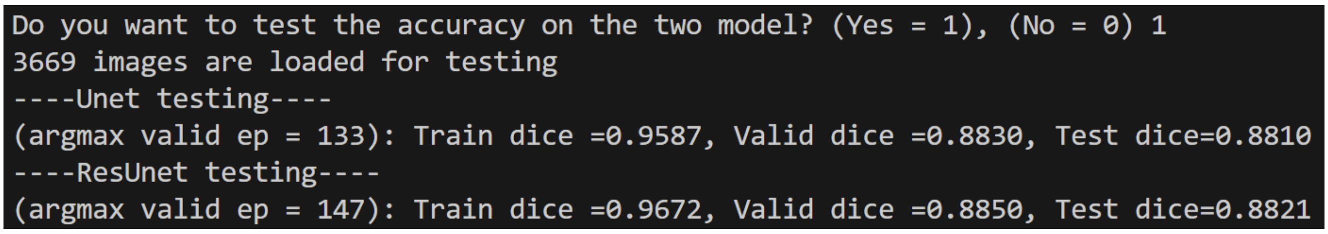


Figure 2. The comparative figures of the training. The learning curve of UNet and UNet-ResNet34. (left) shows the result of (batch = 16, lr = 0.001) and (right) shows the result of (batch = 16, lr = 0.0001)

B. Performance prediction

I used the model which performances the best in the validation dataset to predict the result in testing dataset. In the Unet, the chosen model is in the 134th epoch, and the prediction accuracy is 95.87% in the training dataset, 88.30% in the validation dataset and 88.1% in the testing dataset. On the other hand, in the UNet-ResNet34, the chosen model weight is in the 147th epoch, and the prediction accuracy is 96.72% in the training dataset, 88.50% in the validation dataset and 88.21% in the testing dataset. This result shows the UNet-ResNet34 is marginally better than the UNet.



```
Do you want to test the accuracy on the two model? (Yes = 1), (No = 0) 1
3669 images are loaded for testing
----Unet testing----
(argmax valid ep = 133): Train dice =0.9587, Valid dice =0.8830, Test dice=0.8810
----ResUnet testing----
(argmax valid ep = 147): Train dice =0.9672, Valid dice =0.8850, Test dice=0.8821
```

C. Predictive outputs

I randomly select an easy case and a hard case to compare the performance of the model prediction in this practice. In the easy case shown in Figure 3, the fig. 93 in the testing dataset has a clear object compared to the background. Therefore, two models extract the boundaries well even in the 30th epoch. However, in the hard case shown in Figure 4, the fig. 100 in the testing dataset is a dog hugged by someone's arm and the color is in grey style. Hence, the model is hard to distinguish the boundaries in the UNet. Nevertheless, the UNet-ResNet34 shows about 75% dice score in this case, which proves the high effectiveness of the UNet-ResNet34 in the hard figures.

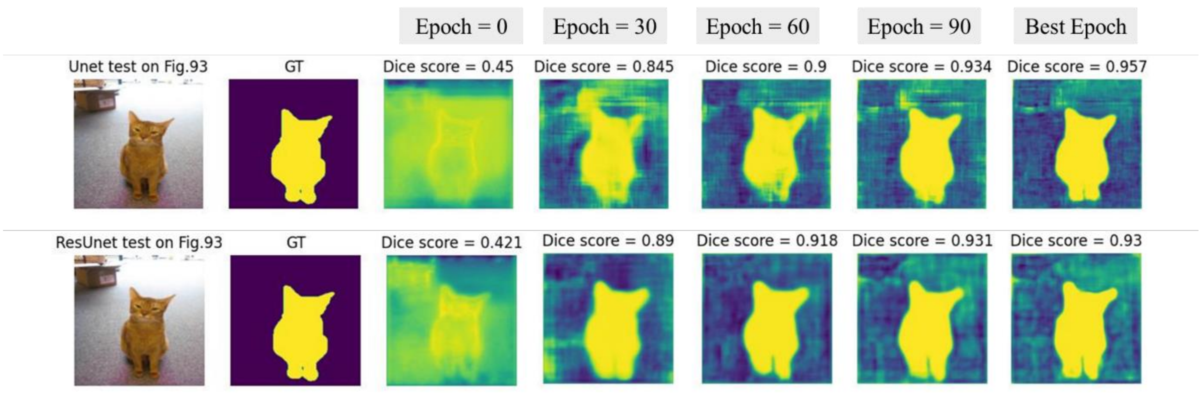


Figure 3. The Fig 93 prediction during the model training.

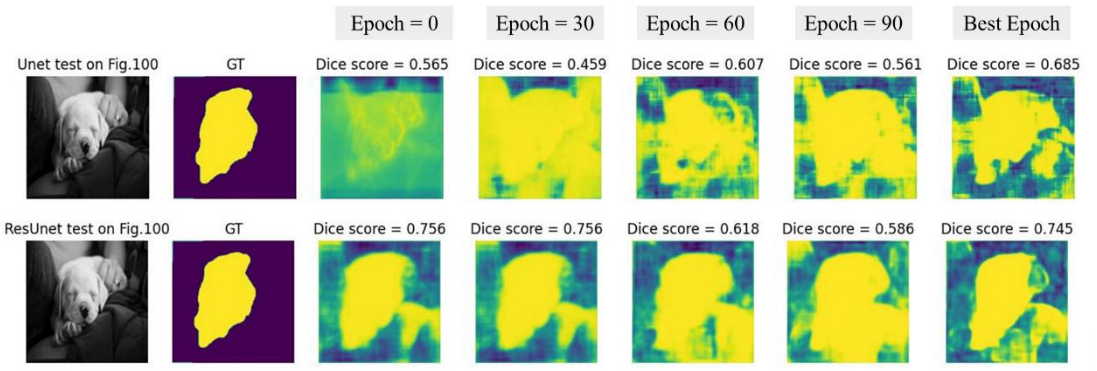
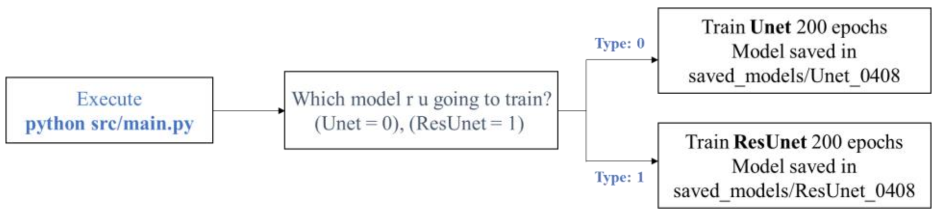


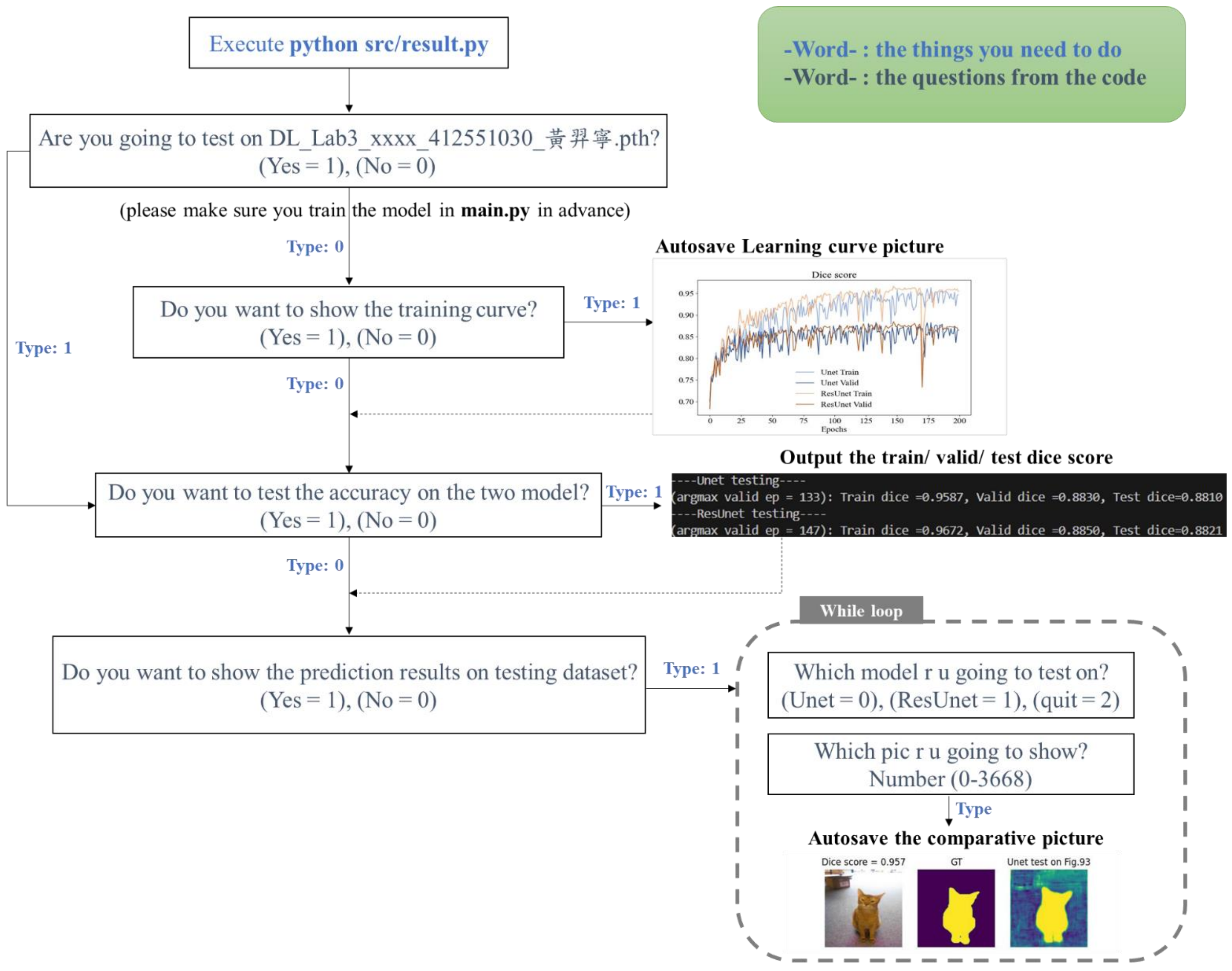
Figure 4. The Fig 100 prediction during the model training

5. Execution command

Training:



Testing:


Execute python src/result.py
-Word- : the things you need to do
-Word- : the questions from the code
Are you going to test on DL_Lab3_xxxx_412551030_黃羿寧.pth?
(Yes = 1), (No = 0)
(please make sure you train the model in main.py in advance)
Type: 0
Type: 1
Do you want to show the training curve?
(Yes = 1), (No = 0)
Type: 1
Autosave Learning curve picture
Dice score
Unet Train
Unet Valid
ResUnet Train
ResUnet Valid
Epochs
Type: 0
Do you want to test the accuracy on the two model?
(Yes = 1), (No = 0)
Type: 1
Output the train/ valid/ test dice score
Type: 0
Do you want to show the prediction results on testing dataset?
(Yes = 1), (No = 0)
Type: 1
While loop
Which model r u going to test on?
(Unet = 0), (ResUnet = 1), (quit = 2)
Which pic r u going to show?
Number (0-3668)
Type
Autosave the comparative picture
Dice score = 0.957
GT
Unet test on Fig.93

```
----Unet testing----
(argmax valid ep = 133): Train dice =0.9587, Valid dice =0.8830, Test dice=0.8810
----ResUnet testing----
(argmax valid ep = 147): Train dice =0.9672, Valid dice =0.8850, Test dice=0.8821
```

6. Discussion

A. What architecture may bring better results?

According to the experimental results in this lab, I found that the two models show similar performance on average, both in training speed and precision of prediction. However, there are still slight differences under certain conditions. Firstly, if my major concern is the speed of training while maintaining a lightweight model, I will choose the UNet for predictions. On the other hand, if my major concern is precision and dealing with complex images, I will select the UNet-ResNet34 due to its superiority. Nevertheless, the tentative conclusion is not entirely reliable since the performance of both models is not outstandingly superior. In the future, I will try data augmentation and use different training parameters for better performance.

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

The semantic segmentation methods are not limited to 2D images in nowadays, but spreading to videos, and 3D data (e.g., fMRI data). There are many applications of this technology. As long as it helps distinguish items of interest from other distractors, this method can be used to solve problems. The potential research topic of the semantic segmentation is spread in various fields, such as medical diagnosis[5, 6], autonomous driving[7], human-machine interaction[8], and surveillance cameras[9]. Tasks that previously required the ability of human brains, such as doctors identifying cancer cells and anomalies, can now significantly reduce the medical burden through this technology. Beyond the application, if research focuses on the techniques of semantic segmentation, according to a review provided by Guo et al. (2018)[10], there is still much room for improvement in model light-weighting and the accuracy of boundary predictions.

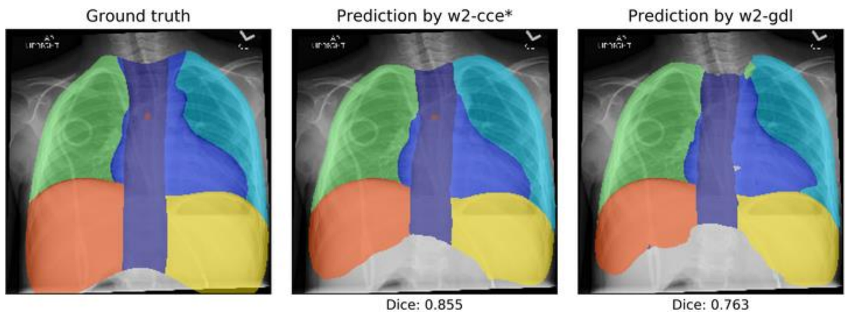


Figure 5 The example of automated multi-region segmentation of pediatric chest radiographs using deep learning[6]

7. References

1 O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, 2015: Springer, pp. 234-241.

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

3 M. Sonka, V. Hlavac, R. Boyle, M. Sonka, V. Hlavac, and R. Boyle, "Image pre-processing," *Image processing, analysis and machine vision,* pp. 56-111, 1993.

4 Y. Zhang and M. A. Mazurowski, "Convolutional neural networks rarely learn shape for semantic segmentation," *Pattern Recognition,* vol. 146, p. 110018, 2024.

5 Y. Wu, L. Lin, J. Wang, and S. Wu, "Application of semantic segmentation based on convolutional neural network in medical images," *Sheng Wu Yi Xue Gong Cheng Xue Za Zhi= Journal of Biomedical Engineering= Shengwu Yixue Gongchengxue Zazhi,* vol. 37, no. 3, pp. 533-540, 2020.

6 G. Holste, R. Sullivan, M. Bindschadler, N. Nagy, and A. Alessio, "Multi-class semantic segmentation of pediatric chest radiographs," in *Medical Imaging 2020: Image Processing*, 2020, vol. 11313: SPIE, pp. 323-330.

7 A. Ess, T. Müller, H. Grabner, and L. Van Gool, "Segmentation-Based Urban Traffic Scene Understanding," in *BMVC*, 2009, vol. 1: Citeseer, p. 2.

8 M. Oberweger, P. Wohlhart, and V. Lepetit, "Hands deep in deep learning for hand pose estimation," *arXiv preprint arXiv:1502.06807,* 2015.

9 M. Gruosso, N. Capece, and U. Erra, "Human segmentation in surveillance video with deep learning," *Multimedia Tools and Applications,* vol. 80, no. 1, pp. 1175-1199, 2021.

10 Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks," *International journal of multimedia information retrieval,* vol. 7, pp. 87-93, 2018.