The Chinese University of Hong Kong, Shenzhen
School of Data Science
Group member name(s): Qiyu Zhang
Group member UID(s): 119010436

Semester 1, 2022
Assignment 1

CSC4080: Artificial Intelligence in Medical Imaging and Health

1 Written Problem

Answer to Question 1

Why is segmentation the most important and popular task in the medical imaging field?

Medical image segmentation is usually the first step of most analysis procedures. It is important to separate regions or objects of interest from other parts of the body. If we can define the borders of the ROI (segment the image) we can often simplify the decisions. It is also a crucial step that determines the final result of the entire application, since the rest of the analysis fully relies on the output from segmentation step.

Answer to Question 2

What are the similarity and differences between classification and segmentation? Explain it from the perspective of data, model, and application.

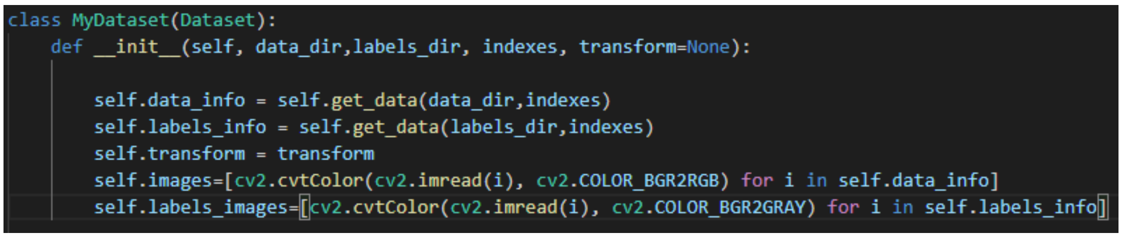
Data: The data all contain original images. The classification method is images and the label list as the excel in our assignment. But the segmentation will have images and masks(The image of the label). The model of the classification: The input is a image, and data augmentation method can be applied. But the output is the chance(no sigmoid) to each label for classification, usually 1d. The output of a segmentation model is a image, usually 1 channel or 3 channel. Application: They are used for various medical imaging are available like MRI, CT, US,positron emission tomography (PET), etc. depending upon need, disease type and body organ. But the classification gives a result or a diagnosis for a kind of disease, while the segmentation separate regions or objects of interest from other parts of the body.

Answer to Question 3

Briefly describe a traditional segmentation method and a deep-learning-based segmentation method. Why does deep learning work better?

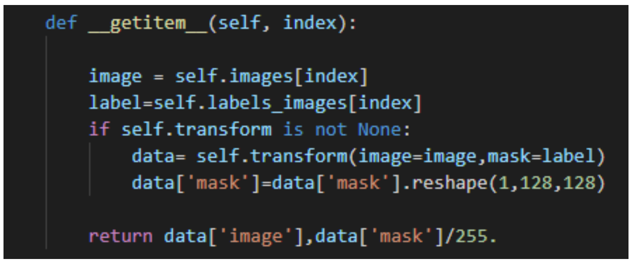
Traditional: Threshold-based segmentation method. calculate one or more grayscale thresholds based on the grayscale features of the image, compare the grayscale value of each pixel in the image with the threshold, and finally classify the pixels into appropriate categories according to the comparison results. Deep learning: Encoder-decoder based models. The model contain the convolution part(encoder) and the deconvolution(decoder) part. The feature is extracted and the mask is generated. The traditional method only consider the characteristics of the pixel gray value itself, but the deep learning consider the spatial characteristics(features) of the images.

2 Coding Problem



```
class MyDataset(Dataset):
    def __init__(self, data_dir,labels_dir, indexes, transform=None):

        self.data_info = self.get_data(data_dir,indexes)
        self.labels_info = self.get_data(labels_dir,indexes)
        self.transform = transform
        self.images=[cv2.cvtColor(cv2.imread(i), cv2.COLOR_BGR2RGB) for i in self.data_info]
        self.labels_images=[cv2.cvtColor(cv2.imread(i), cv2.COLOR_BGR2GRAY) for i in self.labels_info]
```

The figure shows the initial part. The data_info records the path of images. The label_info records the path of labels. Then the images are convert to 3 channels. The labels are converted to 1 channel. Indexes means the indexes of the images and labels that are chosen.

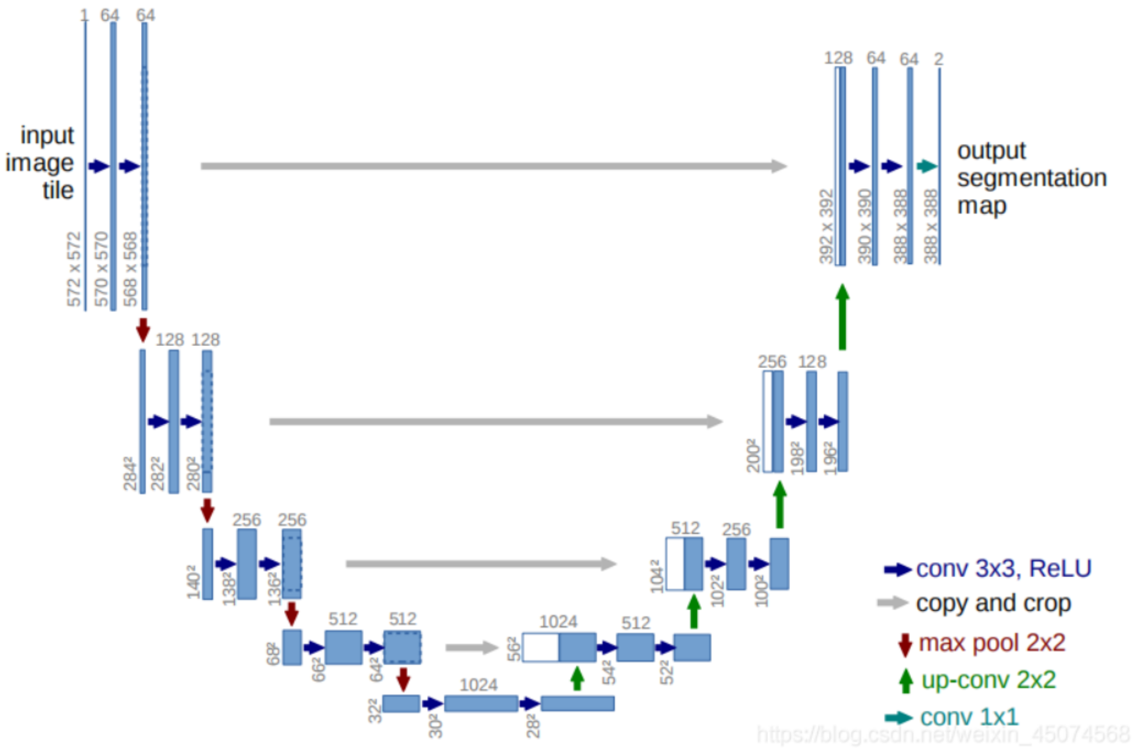


```
def __getitem__(self, index):

    image = self.images[index]
    label=self.labels_images[index]
    if self.transform is not None:
        data= self.transform(image=image,mask=label)
        data['mask']=data['mask'].reshape(1,128,128)

    return data['image'],data['mask']/255.
```

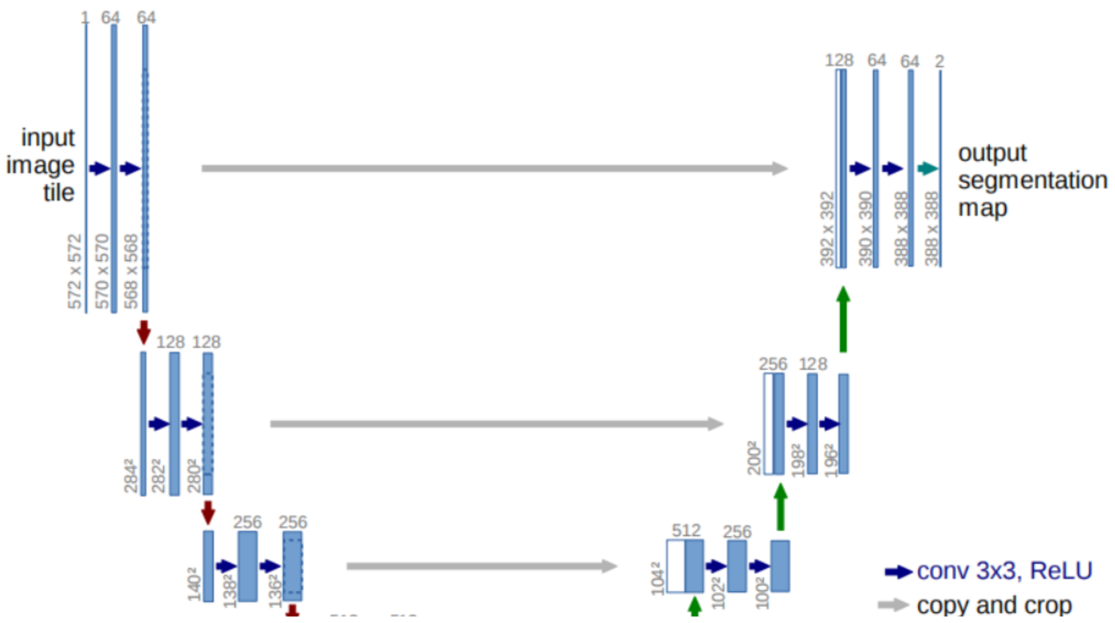
We transform the original image and the label at the same time. The shapes of the images and the masks are in the shape of 3*128*128 and 128*128. To make the labels fit the output, we reshape the labels to 1*128*128.

2.1 Model

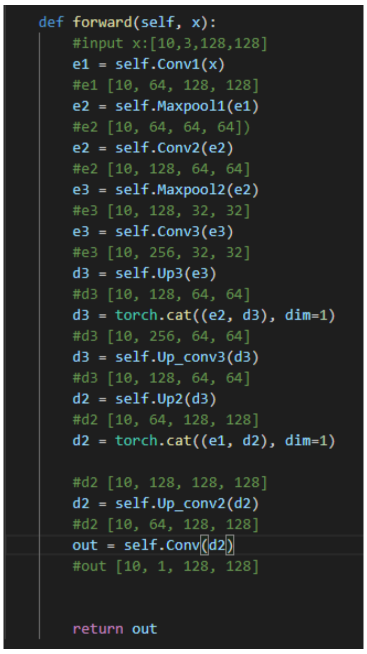
The iuput shape of the Unet is 10*3*128*128. The output shape is 10*1*128*128 (the same with the labels in a batch). The '10' means the batch size,and the '3' and '1' means input and out put channel, '128*128' is the size of an image or label. The model is based on UNet:



We simplify it to the network below:



That means we cut down the last 2 layers. The most difficult thing is that we should change the final output to 1 channel. It can be done if we change the shape of the last convolution layer to fit the shape the the output. The demension of the feature maps are:



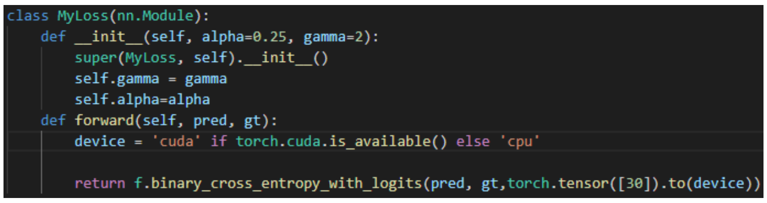
```
def forward(self, x):
    #input x:[10,3,128,128]
    e1 = self.Conv1(x)
    #e1 [10, 64, 128, 128]
    e2 = self.Maxpool1(e1)
    #e2 [10, 64, 64, 64])
    e2 = self.Conv2(e2)
    #e2 [10, 128, 64, 64]
    e3 = self.Maxpool2(e2)
    #e3 [10, 128, 32, 32]
    e3 = self.Conv3(e3)
    #e3 [10, 256, 32, 32]
    d3 = self.Up3(e3)
    #d3 [10, 128, 64, 64]
    d3 = torch.cat((e2, d3), dim=1)
    #d3 [10, 256, 64, 64]
    d3 = self.Up_conv3(d3)
    #d3 [10, 128, 64, 64]
    d2 = self.Up2(d3)
    #d2 [10, 64, 128, 128]
    d2 = torch.cat((e1, d2), dim=1)

    #d2 [10, 128, 128, 128]
    d2 = self.Up_conv2(d2)
    #d2 [10, 64, 128, 128]
    out = self.Conv(d2)
    #out [10, 1, 128, 128]


    return out
```

2.2 loss function

The weighted loss is applied to escape the problem of the unbalanced data. The inputs are the outputs and labels (labels are all '0' or '1'). For each image and label, we flatten it into a 1d array. And apply the sigmoid function to the output. The formula is:

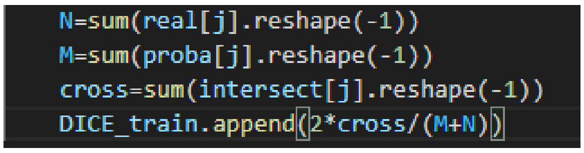
$$loss = -\sum 30ylogp + (1-y)log(1-p)$$

. That means the positive pixels has 30 times weight of the negative pixels. This method encourages the model to give a higher probability to the positive pixels. The code is:



```python
class MyLoss(nn.Module):
    def __init__(self, alpha=0.25, gamma=2):
        super(MyLoss, self).__init__()
        self.gamma = gamma
        self.alpha=alpha
    def forward(self, pred, gt):
        device = 'cuda' if torch.cuda.is_available() else 'cpu'

        return f.binary_cross_entropy_with_logits(pred, gt,torch.tensor([30]).to(device))
```

3 Dice score



```python
N=sum(real[j].reshape(-1))
M=sum(proba[j].reshape(-1))
cross=sum(intersect[j].reshape(-1))
DICE_train.append(2*cross/(M+N))
```

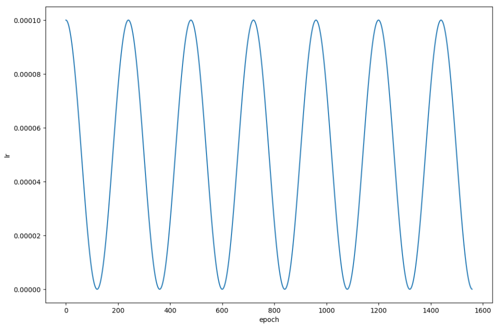
To briefly calculate the dice score, we first extract each image from the whole batch. The proba is a matrix from the sigmoid function of outputs. real is the label images. The dice is defined as:

$$DICE = \frac{2 \times sum(real \cdot proba)}{sum(real) + sum(proba)}$$

To make the calculation faster, $sum(real \cdot proba)$ is seen as the overlap region.

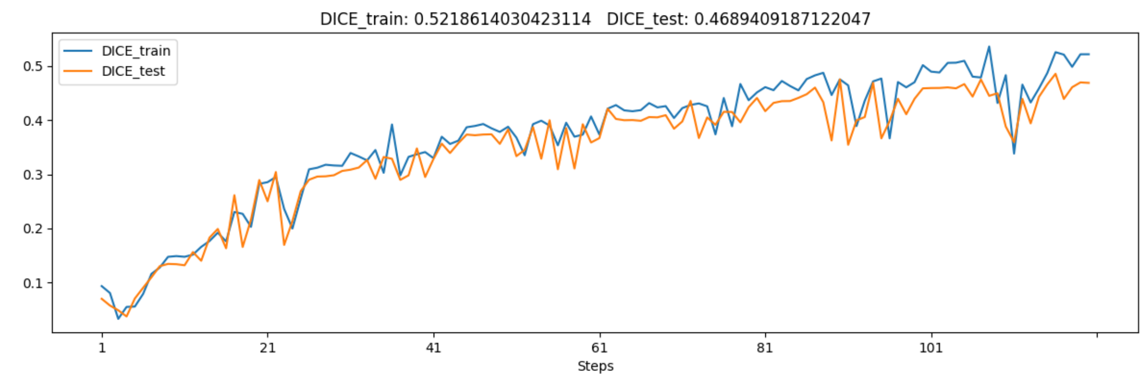
4 Cycling learning rate

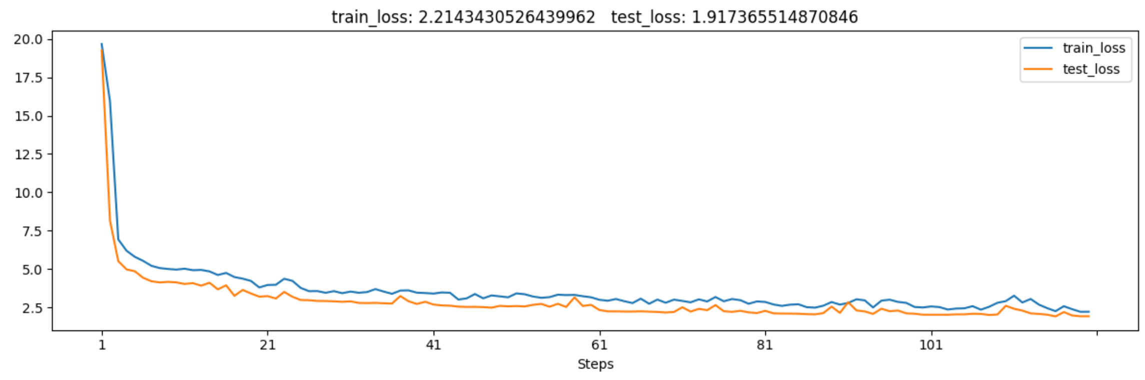
The cosine learning rate is applied. And the visualized learning rate in each folders (5 folder for cross validation, the first one is chosen) is below:(The xlabel means the total number of learning rates used, 120 epochs and 13 batches in each epoch, so there are total 13*120 learning rates)



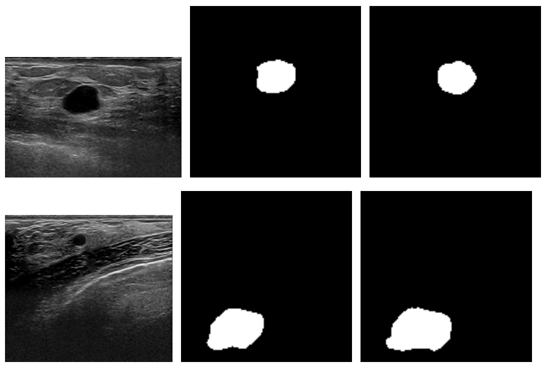
5 Visualized Graph

The figure below shows the Dice efficient and the loss corresponding to the first folder:



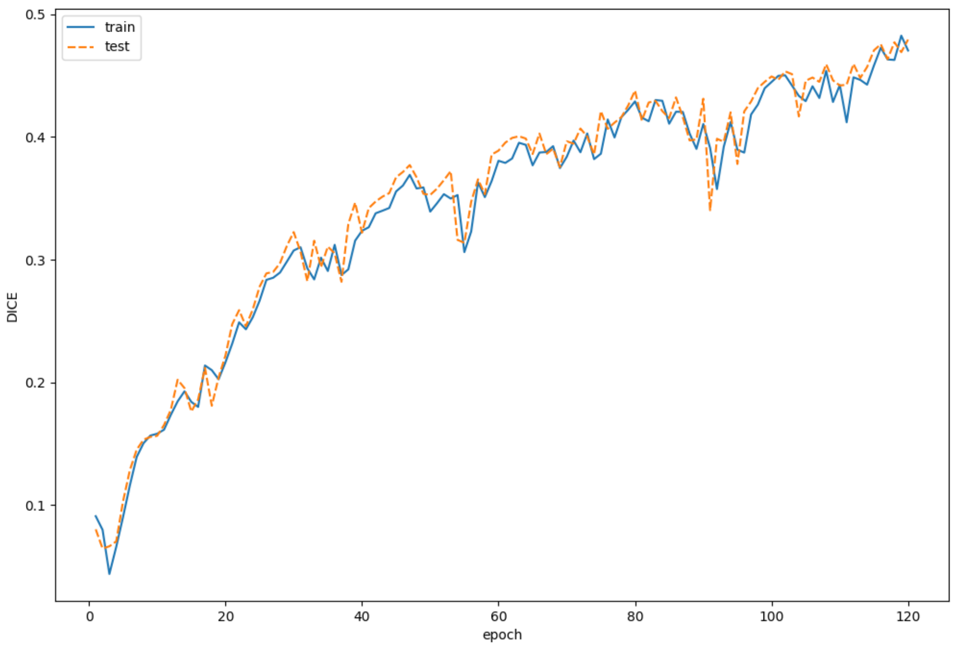


The original figure, the label after data augmentation,and the output labels in the first batch are:



Here only the second and third indexes are shown. The first row is the second original images, mask and label. The second row is the third one (Run the program or open the images in the folder to get more). It can been seen that the model is efficient since that prediction is good. But more epochs can be applied since the tendency of the dice is increasing. The average maximum dice in the cross validation is over 0.5.

The average accuracy of 5 cross validations is:


train
test
DICE
epoch
0.1
0.2
0.3
0.4
0.5
0
20
40
60
80
100
120