2024.01.04

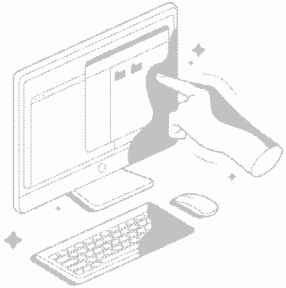
Segment vasculature in 3D scans of human kidney

김예은 원준혁 임종우

CV	Detection & Segmentation	KUBIG

contents

DETECTION &
SEGMENTATION



대회 소개 >

#의료데이터 #Segmentation

[Overview]

- 대회 이름: SenNet + HOA - Hacking the Human Vasculature in 3D
- 목표: HIP-CT로 촬영한 3D Kidney image에서 주어진 혈관 이미지 데이터를 효과적으로 segmentation 하는 것

[Description]

- VCCF는 인체의 혈관 구조를 주요 네비게이션 시스템으로 활용하여 세포들을 매핑함. 이 프레임워크는 모든 규모 수준을 관통하며, 모세혈관 구조를 주소로 활용하여 세포 위치를 식별하는 독특한 방법을 제공하고 있음.
- 그러나 연구자들이 혈관 구조에 대해 알지 못하는 부분으로 인해 VCCF에도 빈틈이 생기고 있기에 이 프로젝트를 통해 혈관 구조의 자동 분할이 가능해진다면, 현실 조직 데이터를 활용하여 이러한 빈틈을 메우고 신체 전체의 혈관 구조를 완성할 것으로 기대할 수 있음.

데이터 소개

DATA SET : 여러 개의 신장에 대한 고해상도 3D 이미지와 그들의 혈관 구조에 대한 3D 분할 마스크
TEST SET : 신장 데이터셋들에 대한 분할 마스크를 생성하는 것이 목표

〈 kidney_1 〉

- kidney_1_dense : 50um 해상도의 우측 신장 전체
- kidney_1_vol : kidney_1의 고해상도 하위 데이터셋(5.2um 해상도)

〈 kidney_2〉

- kidney_2 : 다른 기부자의 신장 전체로 65% 미세 분할 된 데이터 셋 (50um 해상도)

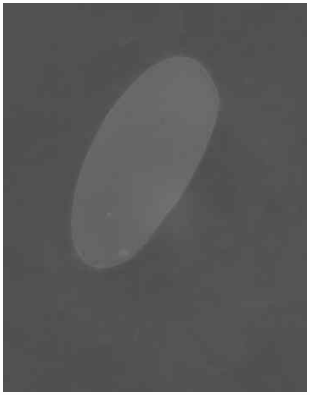
〈 kidney_3 〉

- kidney_3_dense : BM05를 사용하여 50.16um 해상도로 신장 일부 (500 슬라이스)
- kidney_3_sparse : kidney_3의 남은 분할 마스크 (약 85%)

데이터 2D 시각화

〈 kidney_1 〉

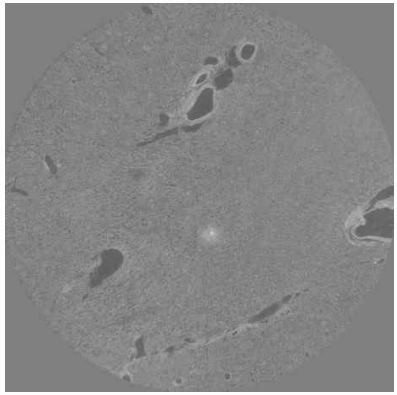
〈 kidney_1_voi 〉



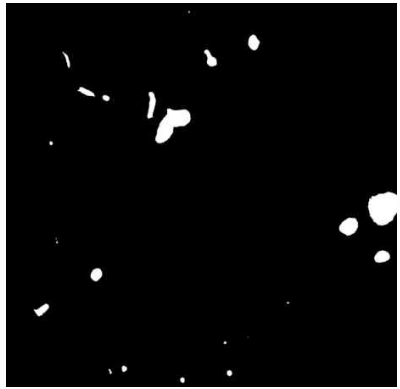
image

label

〈 kidney_1_dense 〉



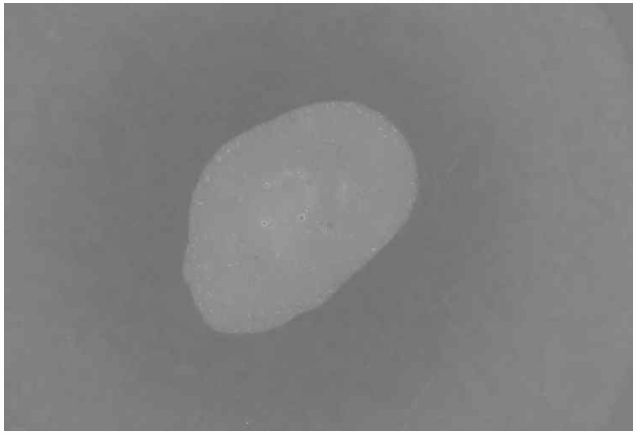
image



label

데이터 2D 시각화

〈 kidney_2 〉



image

label

데이터 2D 시각화

〈 kidney_3 〉

〈 kidney_3_dense 〉

label

〈 kidney_3_sparse 〉

image

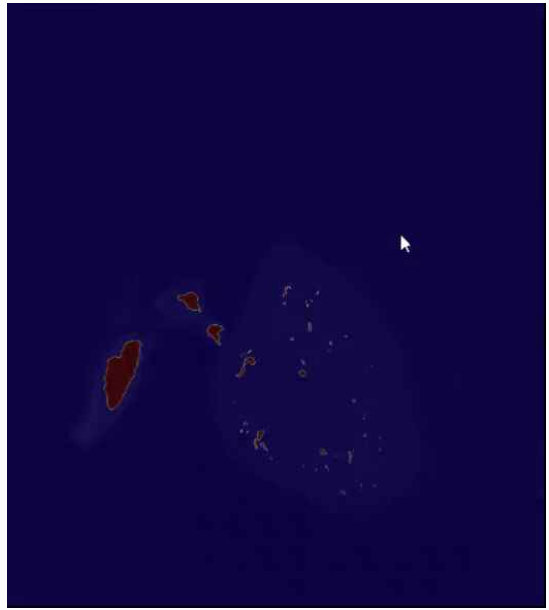
label

데이터 2D 시각화

→ 학습 데이터 중 일부(kidney 3)를 순서대로 모두 표시한 것.

→ 밝게 빛나는 부분이 정답 mask.

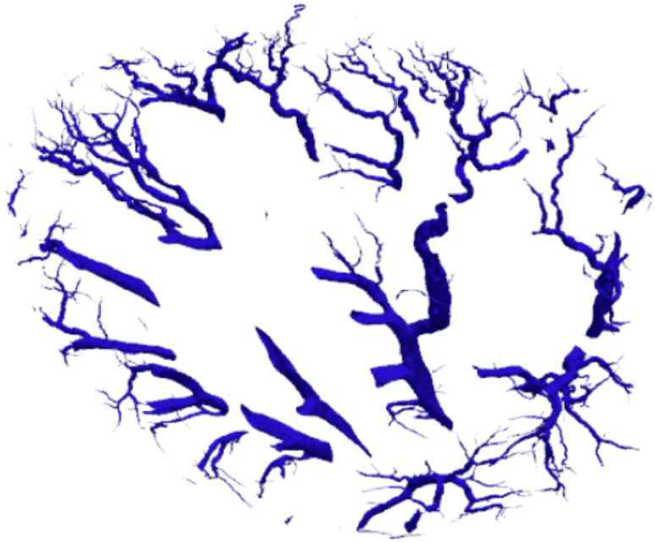
→ 3D kidney CT image를 slicing 한 것.

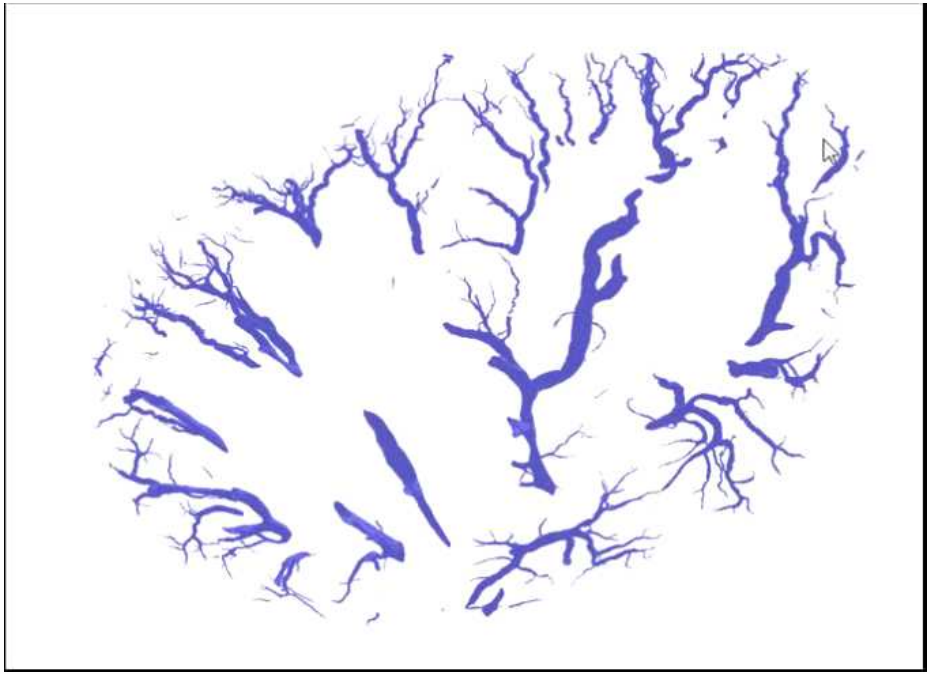


데이터 3D 시각화

→ mask(label)를 stack하여 plotly로 시각화한 결과

→ 데이터의 특성 및 구성을 시각화하여 파악에 용이함.





Attention U-Net

두가지 큰 CT 복부 데이터셋의 multi-class image segmentation으로 평가받는 Attention U-Net 아키텍처

→ 각 skip connection에 대한 게이팅 신호는 여러 이미징 크기에서 정보를 집계하여 쿼리 신호의 그리드 해상도를 높이고 더 나은 성능을 달성하도록 함.

→ AG 파라미터들은 hard-attention에서 사용되는 샘플링 기반의 업데이트하는 방법 없이도 표준 역전파 업데이트로 훈련 가능.

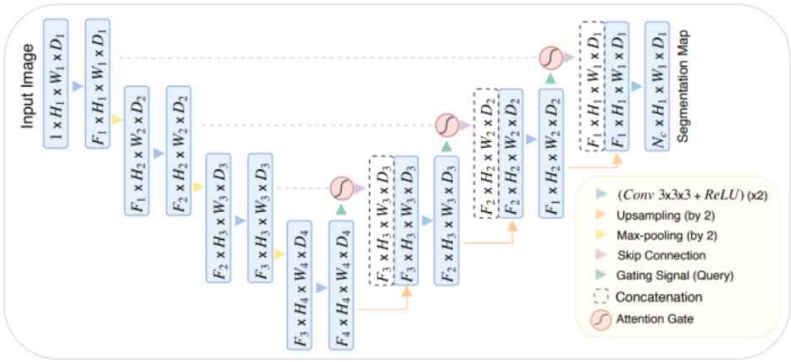
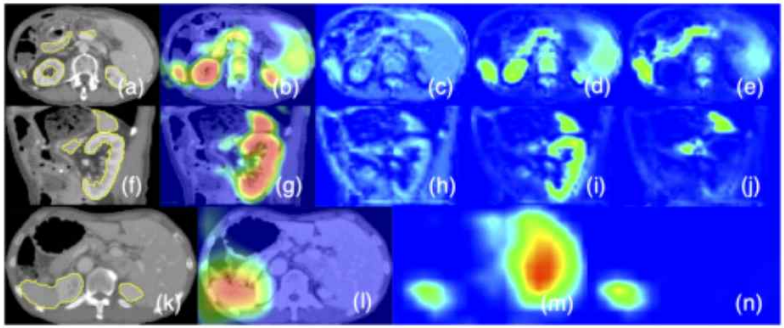
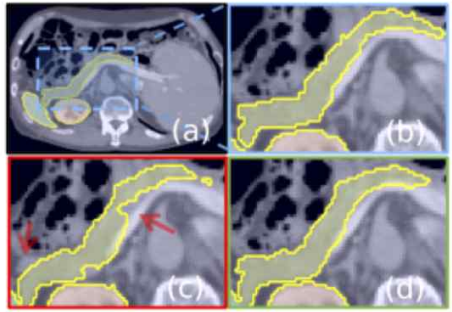


Figure 1: A block diagram of the proposed Attention U-Net segmentation model. Input image is progressively filtered and downsampled by factor of 2 at each scale in the encoding part of the network (e.g. $H_4 = H_1/8$). N_c denotes the number of classes. Attention gates (AGs) filter the features propagated through the skip connections. Schematic of the AGs is shown in Figure 2. Feature selectivity in AGs is achieved by use of contextual information (gating) extracted in coarser scales.

Attention U-Net

두가지 큰 CT 복부 데이터셋의 multi-class image segmentation으로 평가받는 Attention U-Net 아키텍처





Attention U-Net

두가지 큰 CT 복부 데이터셋의 multi-class image segmentation으로 평가받는 Attention U-Net 아키텍처

Method (Train/Test Split)	U-Net (120/30)	Att U-Net (120/30)	U-Net (30/120)	Att U-Net (30/120)
Pancreas DSC	0.814±0.116	**0.840±0.087**	0.741±0.137	**0.767±0.132**
Pancreas Precision	0.848±0.110	0.849±0.098	0.789±0.176	**0.794±0.150**
Pancreas Recall	0.806±0.126	**0.841±0.092**	0.743±0.179	**0.762±0.145**
Pancreas S2S Dist (mm)	2.358±1.464	**1.920±1.284**	3.765±3.452	3.507±3.814
Spleen DSC	0.962±0.013	0.965±0.013	0.935±0.095	**0.943±0.092**
Kidney DSC	0.963±0.013	0.964±0.016	0.951±0.019	0.954±0.021
Number of Params	5.88 M	6.40 M	5.88 M	6.40 M
Inference Time	0.167 s	0.179 s	0.167 s	0.179 s

여러 CNN 모델 없이도 sota를 달성하면서 다양한 데이터 세트와 훈련 크기에 걸쳐 예측 정확도를 지속적으로 개선한다는 것을 보여줌.

2D MODEL CODE

```python
class AttentionBlock(nn.Module):
    """Attention block with learnable parameters"""

    def __init__(self, F_g, F_l, n_coefficients):
        """
        :param F_g: number of feature maps (channels) in previous layer
        :param F_l: number of feature maps in corresponding encoder layer, transferred via skip connection
        :param n_coefficients: number of learnable multi-dimensional attention coefficients
        """
        super(AttentionBlock, self).__init__()

        self.W_gate = nn.Sequential(
            nn.Conv2d(F_g, n_coefficients, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(n_coefficients)
        )

        self.W_x = nn.Sequential(
            nn.Conv2d(F_l, n_coefficients, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(n_coefficients)
        )

        self.psi = nn.Sequential(
            nn.Conv2d(n_coefficients, 1, kernel_size=1, stride=1, padding=0, bias=True),
            nn.BatchNorm2d(1),
            nn.Sigmoid()
        )

        self.relu = nn.ReLU(inplace=True)

    def forward(self, gate, skip_connection):
        """
        :param gate: gating signal from previous layer
        :param skip_connection: activation from corresponding encoder layer
        :return: output activations
        """
        g1 = self.W_gate(gate)
        x1 = self.W_x(skip_connection)
        psi = self.relu(g1 + x1)
        psi = self.psi(psi)
        out = skip_connection * psi
        return out
```

2D MODEL CODE

```
class AttentionUNet(nn.Module):

    def __init__(self, img_ch=3, output_ch=1):
        super(AttentionUNet, self).__init__()

        self.MaxPool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.Conv1 = ConvBlock(img_ch, 64)
        self.Conv2 = ConvBlock(64, 128)
        self.Conv3 = ConvBlock(128, 256)
        self.Conv4 = ConvBlock(256, 512)
        self.Conv5 = ConvBlock(512, 1024)

        self.Up5 = UpConv(1024, 512)
        self.Att5 = AttentionBlock(F_g=512, F_l=512, n_coefficients=256)
        self.UpConv5 = ConvBlock(1024, 512)

        self.Up4 = UpConv(512, 256)
        self.Att4 = AttentionBlock(F_g=256, F_l=256, n_coefficients=128)
        self.UpConv4 = ConvBlock(512, 256)

        self.Up3 = UpConv(256, 128)
        self.Att3 = AttentionBlock(F_g=128, F_l=128, n_coefficients=64)
        self.UpConv3 = ConvBlock(256, 128)

        self.Up2 = UpConv(128, 64)
        self.Att2 = AttentionBlock(F_g=64, F_l=64, n_coefficients=32)
        self.UpConv2 = ConvBlock(128, 64)

        self.Conv = nn.Conv2d(64, output_ch, kernel_size=1, stride=1, padding=0)
```

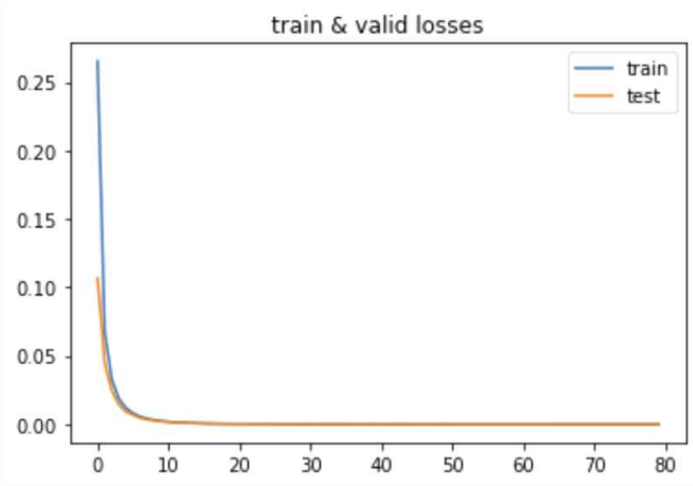
2D MODEL CODE

```
def forward(self, x):
    """
    e : encoder layers
    d : decoder layers
    s : skip-connections from encoder layers to decoder layers
    """
    e1 = self.Conv1(x)

    e2 = self.MaxPool(e1)
    e2 = self.Conv2(e2)

    e3 = self.MaxPool(e2)
    e3 = self.Conv3(e3)

    e4 = self.MaxPool(e3)
    e4 = self.Conv4(e4)

    e5 = self.MaxPool(e4)
    e5 = self.Conv5(e5)

    d5 = self.Up5(e5)

    s4 = self.Att5(gate=d5, skip_connection=e4)
    d5 = torch.cat((s4, d5), dim=1)
    d5 = self.UpConv5(d5)

    d4 = self.Up4(d5)
    s3 = self.Att4(gate=d4, skip_connection=e3)
    d4 = torch.cat((s3, d4), dim=1)
    d4 = self.UpConv4(d4)

    d3 = self.Up3(d4)
    s2 = self.Att3(gate=d3, skip_connection=e2)
    d3 = torch.cat((s2, d3), dim=1)
    d3 = self.UpConv3(d3)

    d2 = self.Up2(d3)
    s1 = self.Att2(gate=d2, skip_connection=e1)
    d2 = torch.cat((s1, d2), dim=1)
    d2 = self.UpConv2(d2)

    out = self.Conv(d2)

    return out
```

RESULT

2D MODEL

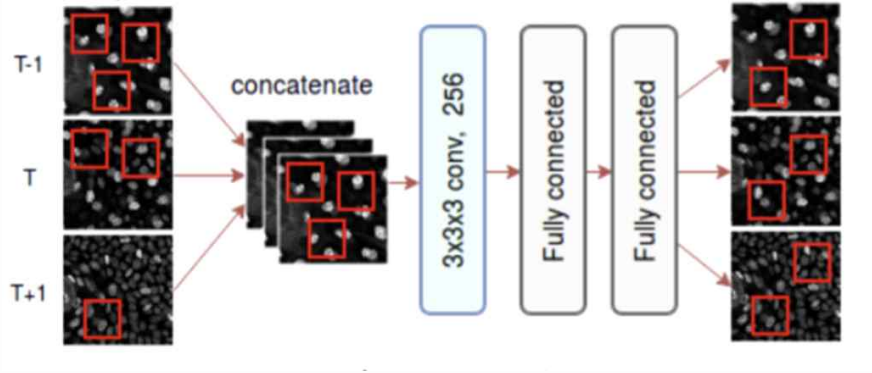
학습 Epoch 별 Score	
25	0.106
45	0.134
65	0.234
80	0.234



2.5D CNN

→ 2D CNN의 경우, RGB 3 Channel을 가지는 image를 convolution network의 input으로 사용함.

→ 본 대회를 진행하며 사용한 2.5D CNN의 경우, 같은 폴더(동일 kidney)에 존재하는 Gray 1 Channel image들을 모두 concat하여 convolution network의 input으로 사용하는 방식



CODE의 차이점

2D MODEL

```
img = cv2.imread(path, cv2.IMREAD_UNCHANGED)
img = np.tile(img[...,None], [1, 1, 3]) # gray to rgb
```

→ 3 Channel을 가지는 image로 사용함

2.5D MODEL

```
img=cv2.imread(self.paths[index],cv2.IMREAD_GRAYSCALE)

def load_data(path,s):
   # 중간 생략 (Dataloader)
    for x in tqdm(data_loader):
        data.append(x)
    return tc.cat(data,dim=0)
```

→ Gray 1 channel의 image들을 concat한 형태로 데이터를 사용하여 학습 진행

RESULT

2.5D MODEL

Backbone MODEL

model	epoch	score
resnext50	19	0.541
resnext101	16	0.555
resnext50	15	0.496

RESULT

2.5D MODEL

Learning rate

model	epoch	lr	score
resnext101	16	0.00006	0.555
resnext101	16	0.0006	0.614
resnext101	19	0.0006	0.618
resnext101	19	0.001	0.554

RESULT

2.5D MODEL

random 확률

model	epoch	lr	p	score
resnext101	19	0.0006	initial	0.618
resnext101	19	0.0006	increase	0.527

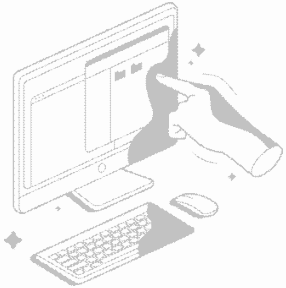
RESULT

2.5D MODEL

최종 결과

model	epoch	lr	p	score
resnext101	19	0.0006	initial	0.618
resnext101	16	0.0006	initial	0.614
resnext101	27	0.0006	initial	0.606
resnext101	23	0.0006	initial	0.591
resnext101	16	0.00006	initial	0.555
resnext101	19	0.001	initial	0.554
resnext50	19	0.00006	initial	0.541
resnext101	19	0.0006	increase	0.527
resnext50	15	0.00006	initial	0.496

→ 여러 시도 끝에 public score 0.618 달성



어려웠던 점

[오류 디버깅]

대회가 Inference notebook을 만들어 제출하면 서버에 있는 추가 데이터로 채점을 진행하는 방식으로 이루어졌는데, 이와 관련해 초반에 오류를 디버깅하는 것이 어려웠다.

[2.5D 추가 모델 학습]

2.5D 모델에서 unet 모델을 먼저 사용해보고 성능이 더 좋은 attention unet 모델을 활용하려 했으나 코드 적용 시 오류가 발생하여 이를 해결하지 못해서 아쉬웠다.

[시각화]

train 시간도 오래 걸렸지만, rle 인코딩 값을 비교하는 특성 상 예측 결과의 public score를 확인하는데도 너무 오랜 시간이 걸렸고, 그 결과를 시각적으로 확인해보지 못했다.

2024.01.04

감사합니다

CV	김예은 원준혁 임종우	thanks