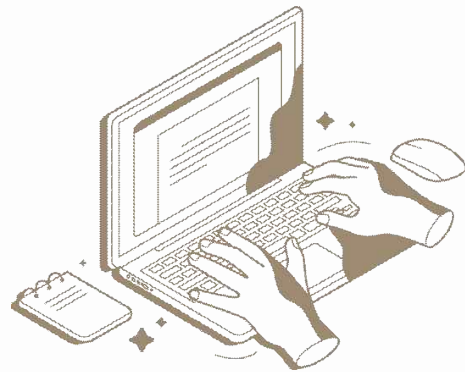


Segment vasculature in 3D scans of human kidney

김예은 원준혁 임종우



CV

Detection & Segmentation

KUBIG

contents

DETECTION &
SEGMENTATION

01

대회 소개

02

데이터 소개 및 시각화

03

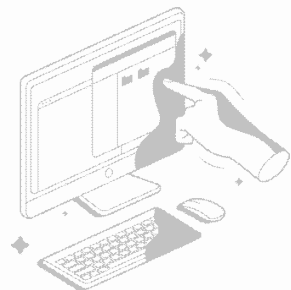
2D 모델 학습 및 예측

04

2.5D 모델 학습 및 예측

05

결과 및 소감



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_voi : 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%)

< Work Flow

01

3D 데이터
시각화 및 탐색

02

2D UNET
Modeling

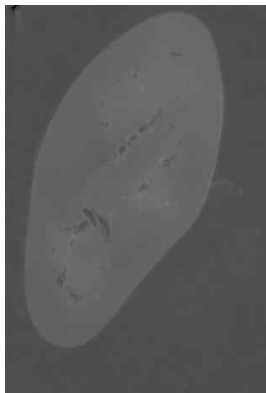
03

2.5D UNET
Modeling

데이터 2D 시각화

< kidney_1 >

< kidney_1_voi >

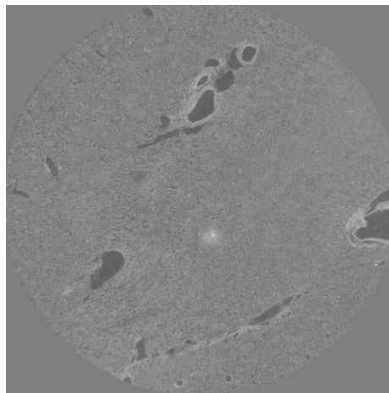


image

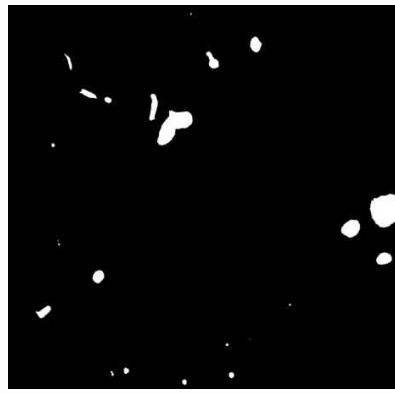


label

< kidney_1_dense >



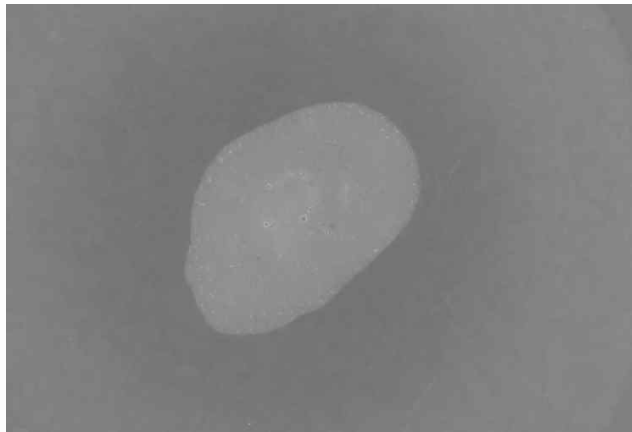
image



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.

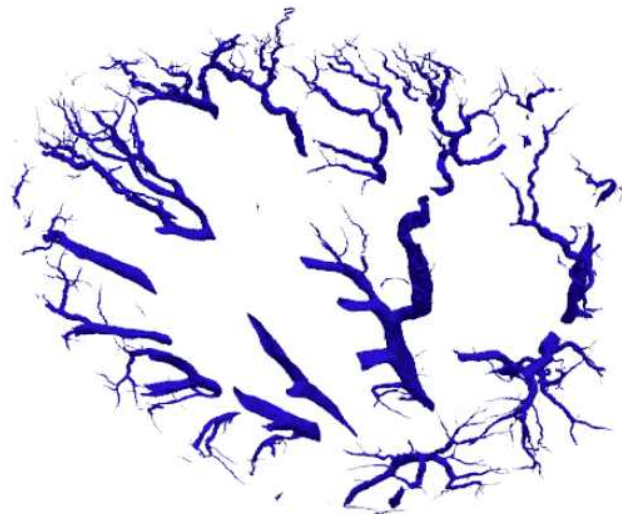
→ 3D kidney CT image를 slicing 한 것.

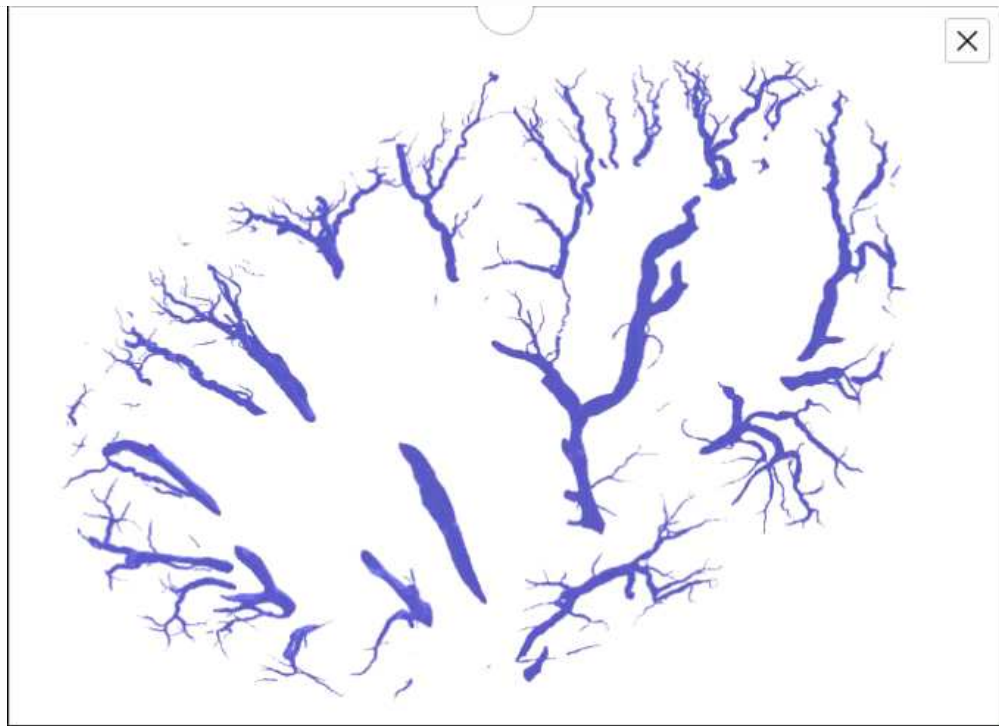


데이터 3D 시각화

→ mask(label)를 stack하여 plotly로 시각화한 결과

→ 데이터의 특성 및 구성을 시각화하여 파악에 용이함.





Attention U-Net

→ 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.

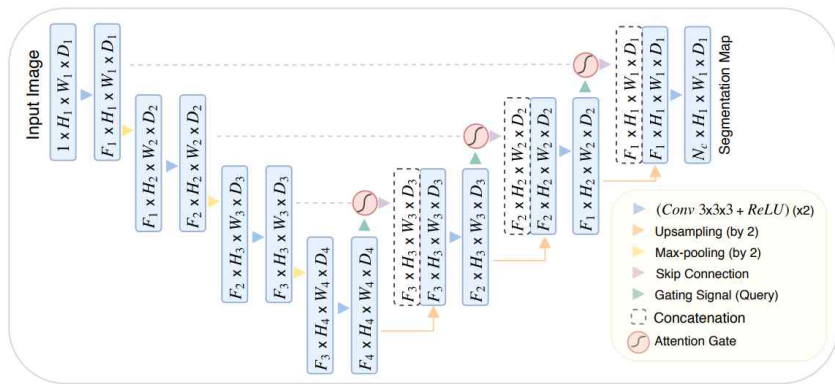
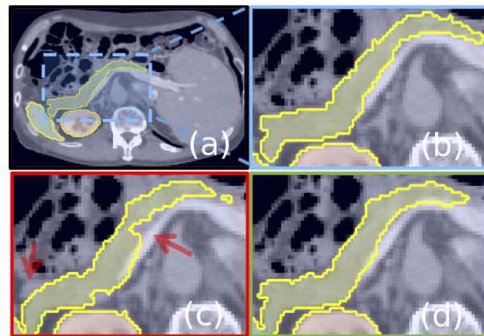
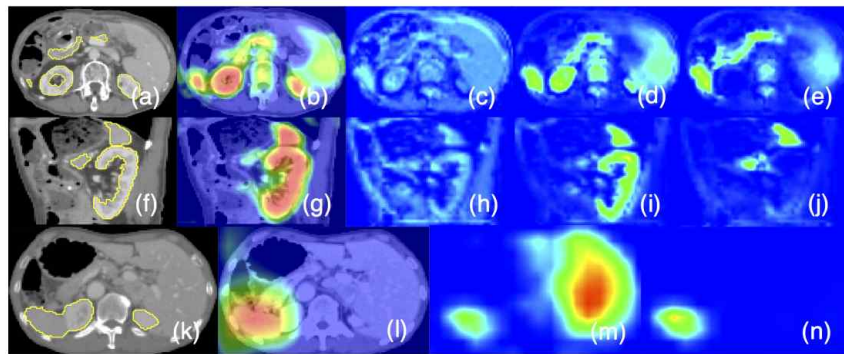


Figure 1: A block diagram of the proposed Attention U-Net segmentation model. Input image is progressively filtered and downsampled by factor of 2 at each scale in the encoding part of the network (e.g. $H_4 = H_1/8$). N_c denotes the number of classes. Attention gates (AGs) filter the features propagated 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



Attention U-Net



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 Attention U-Net

```
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)
```

```
    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)
```

```
        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)
```

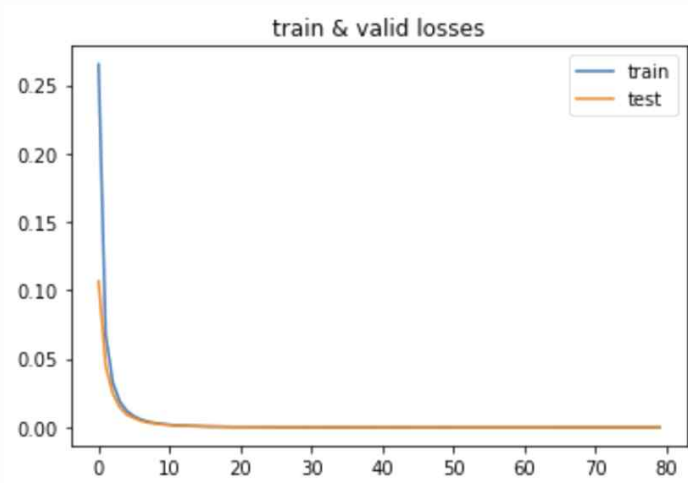
```
        return out
```

RESULT



2D MODEL

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



2D vs 2.5D

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)
```

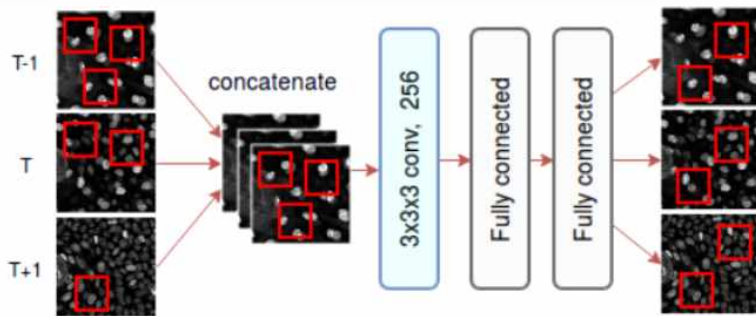
```
def load_data(path,s):
    # 중간 생략 (Dataloader)
    for x in tqdm(data_loader):
        data.append(x)
    return tc.cat(data,dim=0)
```

→ Gray 1 channel의 image들을 concat한
형태로 데이터를 사용하여 학습 진행

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 concatenate convolution network의 input으로 사용하는 방식



출처:

Kitrungrotsakul, Thinnunt, et al. "A 2.5 d cascaded convolutional neural network with temporal information for automatic mitotic cell detection in 4d microscopic images." 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). IEEE, 2018

2.5D MODEL

```
class CFG:
    # ===== pred target =====
    target_size = 1

    # ===== model CFG =====
    model_name = 'Unet'
    backbone = 'se_resnext101_32x4d'

    in_chans = 5 # 입력 채널 수
    # ===== training CFG =====
    image_size = 256
    input_size=256
    drop_egde_pixel = 0
    tile_size = image_size
    stride = tile_size // 2
    assert stride>drop_egde_pixel # stride가 drop_edge_pixel보다 커야 함

    train_batch_size = 16 # 32
    valid_batch_size = train_batch_size * 2

    epochs = 30 # 에폭 수
    lr = 6e-4 # 학습률
```

[Unet 사용자 모델 정의]

model 구조 및 backbone 모델 정의

batch size, learning rate, epoch 등 설정

2.5D MODEL

```
# ===== augmentation =====
train_aug_list = [
    A.RandomResizedCrop(
        input_size, input_size, scale=(0.8,1.25)),
    A.ShiftScaleRotate(p=0.75),
    A.OneOf([
        A.GaussNoise(var_limit=[10, 50]),
        A.GaussianBlur(),
        A.MotionBlur(),
    ], p=0.4),
    A.GridDistortion(num_steps=5, distort_limit=0.3, p=0.5),
    ToTensorV2(transpose_mask=True),
]
train_aug = A.Compose(train_aug_list)
valid_aug_list = [
    ToTensorV2(transpose_mask=True),
]
valid_aug = A.Compose(valid_aug_list)
```

[Augmentation]

1. RandomResizedCrop

- 무작위로 크롭하여 이미지의 크기 조절

2. ShiftScaleRotate

- 무작위로 이미지 이동, 크기 조절, 회전

3. GaussNoise(가우시안 노이즈), GaussianBlur

(가우시안 블러), MotionBlur(모션 블러) 중 하나 선택

4. GridDistortion

- 격자 왜곡을 적용하여 이미지 왜곡

2.5D MODEL

```
def min_max_normalization(x:tc.Tensor)->tc.Tensor:
    """input.shape=(batch,f1,...)"""
    shape=x.shape # 입력 텐서의 형태를 저장
    if x.ndim>2:
        x=x.reshape(x.shape[0],-1) # 입력 텐서의 차원을 하나로 평탄화

    min_=x.min(dim=-1,keepdim=True)[0]
    max_=x.max(dim=-1,keepdim=True)[0]
    if min_.mean()==0 and max_.mean()==1:
        return x.reshape(shape)

    x=(x-min_)/(max_-min_+1e-9) # 픽셀값을 0과 1사이로 조정 , 1e-9는 작은 상수
    return x.reshape(shape)
```

[데이터 정규화]

min max 정규화 : 수치 변수의 값을 조절하여
모든 특성이 표준 스케일에 있도록 설정

2.5D MODEL

```
class Data_loader(Dataset):
    def __init__(self, path, s="/images/"):
        self.paths=glob(path+f"{s}*.tif")
        self.paths.sort()
        self.bool=s=="labels/"

    def __len__(self):
        return len(self.paths)

    def __getitem__(self, index):
        img=cv2.imread(self.paths[index], cv2.IMREAD_GRAYSCALE)
        # 이미지를 그레이스케일로 읽어옴
        img=tc.from_numpy(img)
        if self.bool:
            img=img.to(tc.bool)
        else:
            img=img.to(tc.uint8)
        return img

def load_data(path, s):
    data_loader=Data_loader(path, s)
    data_loader=DataLoader(data_loader, batch_size=16, num_workers=2)
    data=[]
    for x in tqdm(data_loader):
        data.append(x)
    return tc.cat(data, dim=0)
```

[데이터 로딩]

- 이미지를 GRAYSCALE로 읽어오고, 여러 이미지들을 일부만 결합
- train 및 valid 생성 (valid data는 voi1dense 데이터로 고정)

2.5D MODEL

```
def dice_coef(pred:tc.Tensor,target:tc.Tensor,TH=0.5,epsilon=1e-5):  
    if tc.any(pred<0) or tc.any(pred>1):  
        pred=pred.sigmoid()  
    target = target.unsqueeze(1).to(tc.float32)  
    pred = (pred>TH).to(tc.float32)  
    inter = (target*pred).sum()  
    den = target.sum() + pred.sum()  
    dice = ((2*inter+epsilon)/(den+epsilon)).mean()  
    return dice
```

[Dice coefficient]

두 집합 간의 유사성 측정하는 계수

2.5D MODEL

```
def rle_encode(mask):
    pixel = mask.flatten()
    pixel = np.concatenate([[0], pixel, [0]]) # 시작과 끝에 0 추가
    run = np.where(pixel[1:] != pixel[:-1])[0] + 1 # 같지 않은 위치를 true로 나타내는 배열
    run[1::2] -= run[::2]
    rle = ''.join(str(r) for r in run)
    if rle == '':
        rle = '1 1'
    return rle

def normalization(x:tc.Tensor, smooth=1e-7)->tc.Tensor:
    """input.shape=(batch, f1, f2,...)"""
    # [batch, f1, f2]->dim[1, 2]
    dim=list(range(1, x.ndim))
    mean=x.mean(dim=dim, keepdim=True)
    std=x.std(dim=dim, keepdim=True)
    return (x-mean)/(std+smooth)
```

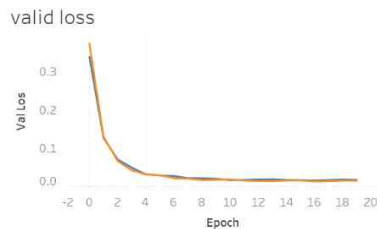
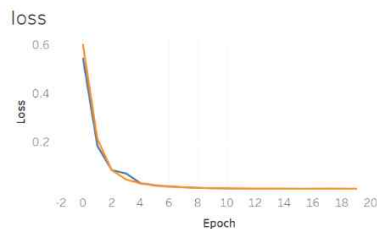
[rle 인코딩 및 정규화]

RESULT

2.5D MODEL

Backbone MODEL

model	epoch	score
resnext50	20	0.541
resnext101	16	0.555
resnext50	15	0.496



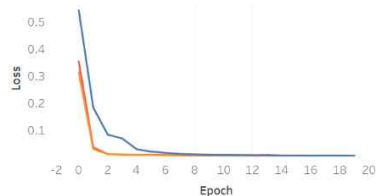
RESULT

2.5D MODEL

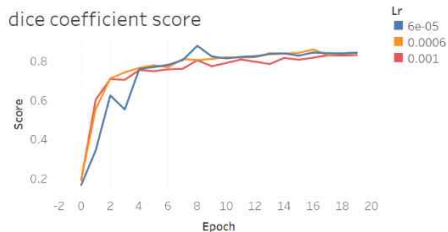
Learning rate

model	epoch	lr	score
resnext101	16	0.00006	0.555
resnext101	16	0.0006	0.614
resnext101	20	0.0006	0.618
resnext101	20	0.001	0.554

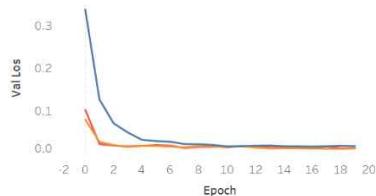
loss



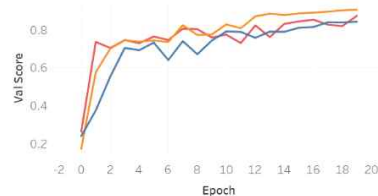
dice coefficient score



valid loss



valid dice coefficient score

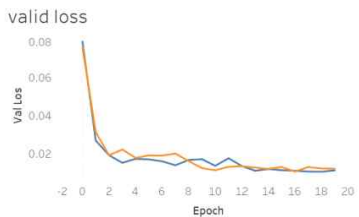
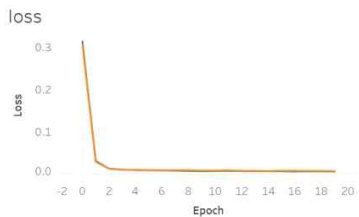


RESULT

2.5D MODEL

random 확률

model	epoch	lr	p	score
resnext101	20	0.0006	initial	0.618
resnext101	19	0.0006	increase	0.527



RESULT



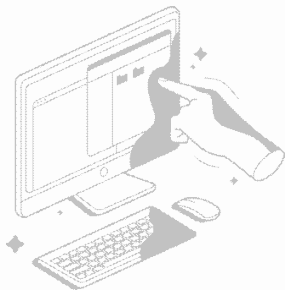
2.5D MODEL

최종 결과



model	epoch	lr	p	score
resnext101	20	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	20	0.001	initial	0.554
resnext50	20	0.00006	initial	0.541
resnext101	20	0.0006	increase	0.527
resnext50	15	0.00006	initial	0.496

→ 여러 시도 끝에 public score 0.618 달성



소감 및 아쉬운 점 >

[2.5D 추가 모델 학습]

2D로 학습을 시킬 때 보다 2.5D를 사용해 여러 이미지를 stack해 한번에 학습함으로써 성능이 크게 증가함을 발견하였다는 점에서 의의가 있었다. 다만, 2.5D 모델에서 unet 모델을 먼저 사용해보고 성능이 더 좋은 attention unet 모델을 활용하려 했으나 코드 적용 시 오류가 발생하여 이를 해결하지 못해서 아쉬웠다.

[오류 디버깅]

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

[시각화]

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

2024.01.04



HAPPY
NEW YEAR



CV

김예은 원준혁 임종우

thanks