(SenNet + HOA - Hacking the Human Vasculature in 3D)

신장 내 혈관 분할

CV SEGMENTATION 1팀
김송성 김연규 최유민

(CONTENTS)

목차

01 Pre-Study
02 대회 및 데이터 소개
03 U-Net
04 3D Inference
05 SCNAS
06 Conclusion

(PRE-STUDY)

Detection & Segmentation

Paper Review

- R-CNN
- YOLO
- U-Net
- SSD
- FPN
- ViT
- SegFormer
- SAM

Toy Project

- 위성 이미지 건물 영역 분할 (데이콘)
- Camera-Invariant Domain Adaptation (데이콘)

Kaggle

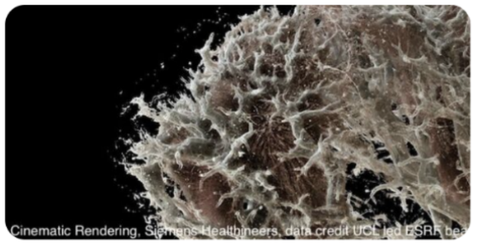
- 3D Segmentation Model 리뷰
- Method for medical domain 리뷰

(OVERVIEW)

대회 소개

SenNet + HOA - Hacking the Human Vasculature in 3D

Segment vasculature in 3D scans of human kidney



3D Hierarchical Phase-Contrast Tomography (HiP-CT)로 촬영된 인체의 신장 데이터에서 **혈관을 분할.**

-> 몸 전체의 혈관 구조 사진을 완성하는 것 도움

-> 인체 조직의 혈관의 크기, 모양, 패턴 등에 대한 연구자의 이해 도움

(OVERVIEW)

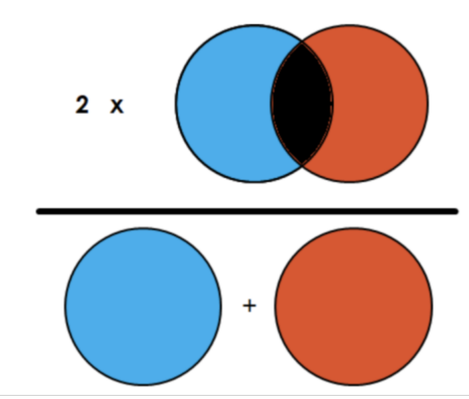
대회 소개

평가지표 | Surface Dice Metric (tolerance : 0.0)

- 두 표면 사이의 겹침을 측정
- 다른 표면과 가장 가까운 거리가 지정된 tolerance보다 작거나 같으면 표면 겹침으로 계산
- 0.0(겹침 없음) ~ 1.0(완전히 겹침)

```
overlap_gt = np.sum(surfel_areas_gt[distances_gt_to_pred <= tolerance_mm])
overlap_pred = np.sum(
    surfel_areas_pred[distances_pred_to_gt <= tolerance_mm])
surface_dice = (overlap_gt + overlap_pred) / (np.sum(surfel_areas_gt) +
                                              np.sum(surfel_areas_pred))
return surface_dice
```

$$Dice = \frac{2 * |A \cap B|}{|A| + |B|} = \frac{2 * TP}{(TP + FP) + (TP + FN)}$$



(DATA)

데이터 소개

- 2D slice of a 3D volume 데이터
- kidney1, 2, 3 세 사람에 대한 image 및 mask
- 각각의 분할된 정보나 비율이 다름

- kidney_5 & kidney_6 : test set

kidney_1

- dense : 50um 해상도의 오른쪽 신장 전체
- voi : 5.2um 해상도에서 kidney_1의 고해상도 부분 집합

kidney_2

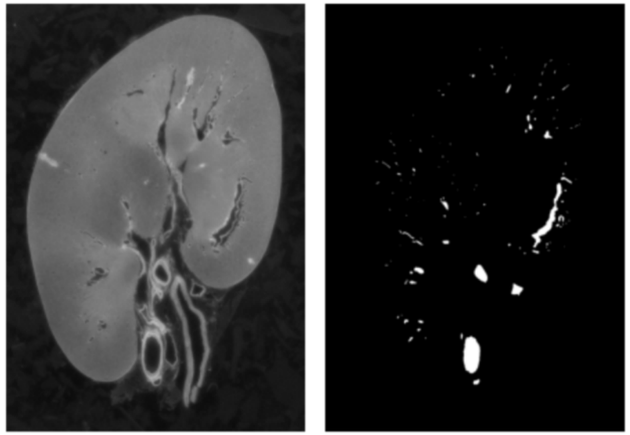
- 50um 해상도의 신장 전체
  Sparsely segmented (about 65%)

kidney_3

- dense : BM05를 사용한 50.16um 해상도의 신장 부분(500 slice)
  Densely segmented
- spare : kidney_3의 나머지 분할 mask
  Sparsely segmented (about 85%).

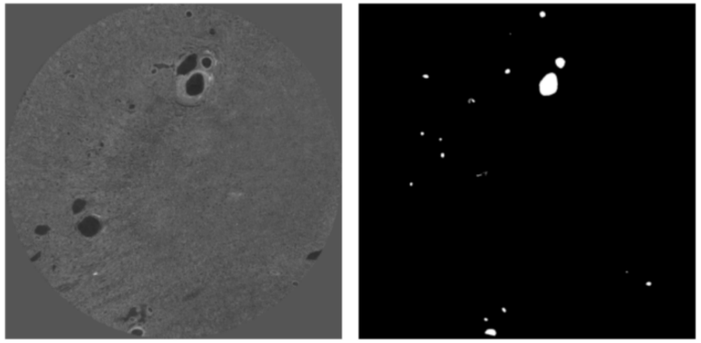
(DATA)

데이터 소개

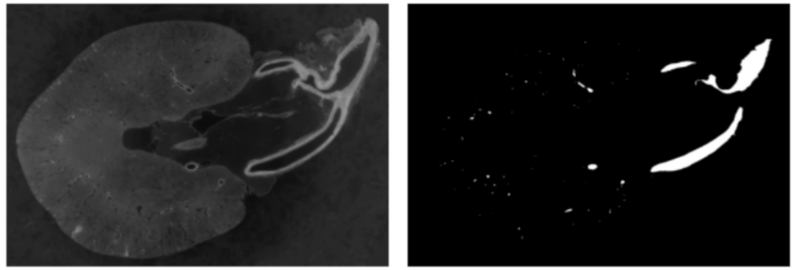


kidney_1_dense
(2279, 1303, 912)

Z X Y



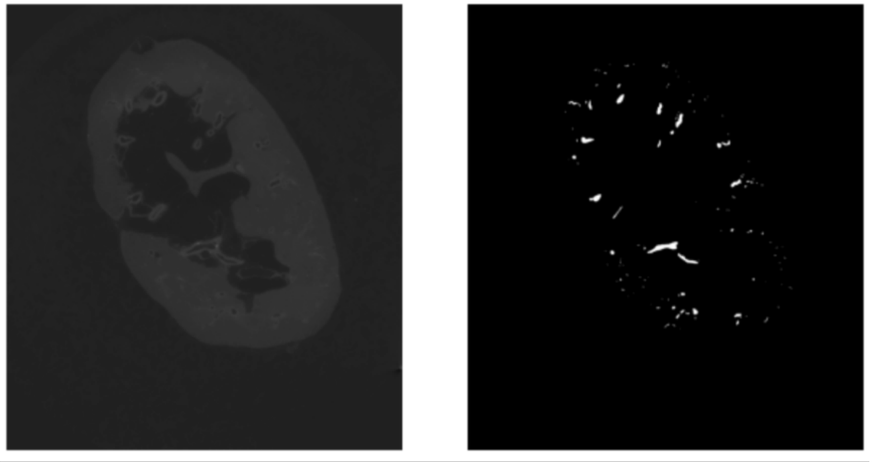
kidney_1_voi
(1397, 1928, 1928)



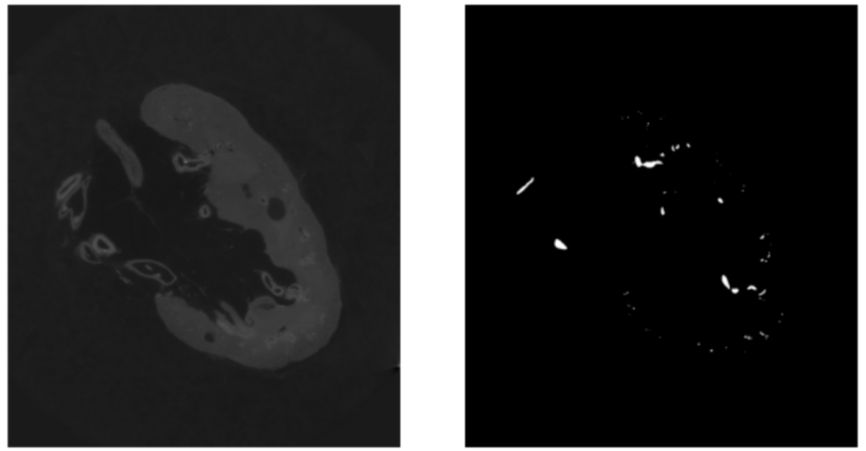
kidney_2
(2217, 1041, 1511)

(DATA)

데이터 소개



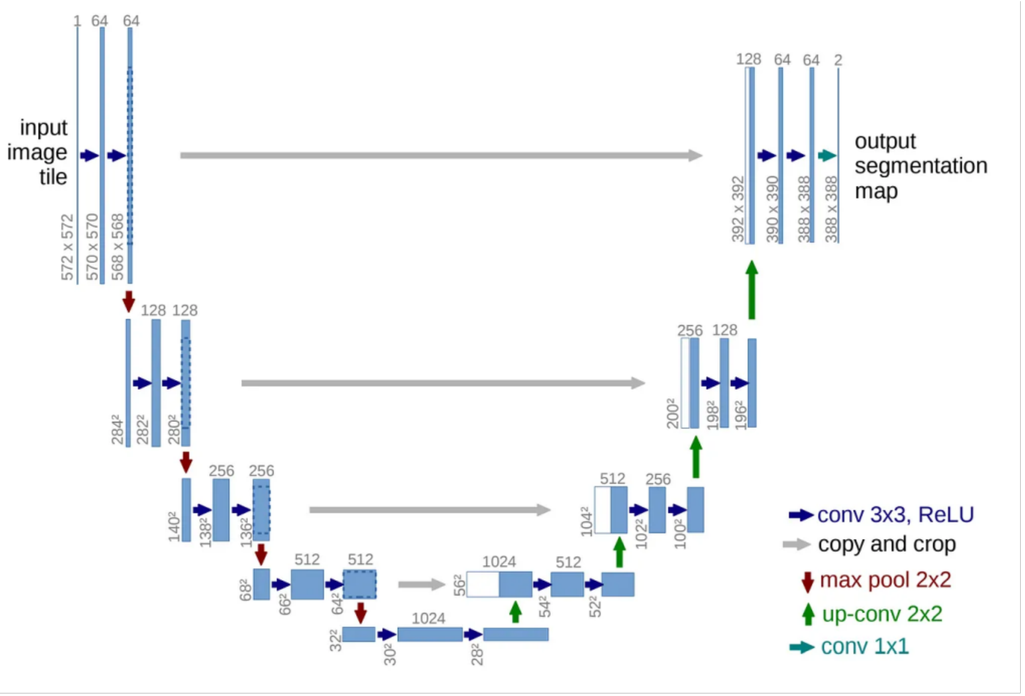
kidney_3_dense
(501, 1706, 1510)



kidney_3_sparse
(1035, 1706, 1510)

(MODELING)

U-Net



Segmentation Models

Segmentation Models Pytorch(SMP) Library
https://github.com/qubvel/segmentation_models.pytorch

```
self.encoder = smp.Unet(
    encoder_name=CFG.backbone,
    encoder_weights=weight,
    in_channels=CFG.in_chans,
    classes=CFG.target_size,
    activation=None,
)
```

- U-Net Architecture
- ResNeXt, SE-Net, DenseNet, EfficientNet 등의 encoder
- pre-trained weights (imagenet / instagram)

(MODELING)

U-Net

2.5d Cutting Model

- CFG.in_chans = 5

```
next(iter(train_dataset))[0].shape
✓ 3.4s
torch.Size([16, 5, 256, 256])
```

```
next(iter(train_dataset))[1].shape
✓ 3.7s
torch.Size([16, 256, 256])
```

```
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 torch.cat(data,dim=0)
```

```
x_index = np.random.randint(0,x.shape[1]-self.image_size)
y_index = np.random.randint(0,x.shape[2]-self.image_size)
```

```
x = x[index:index+self.in_chans,x_index:x_index+self.image_size,y_index:y_index+self.image_size].to(torch.float32)
y = y[index+self.in_chans//2,x_index:x_index+self.image_size,y_index:y_index+self.image_size].to(torch.float32)
```

```
data = self.transform(image=x.numpy().transpose(1,2,0), mask=y.numpy())
x = data['image']
y = data['mask']
```

(MODELING)

U-Net

```
self.encoder = smp.Unet(
    encoder_name=CFG.backbone,
    encoder_weights=weight,
    in_channels=CFG.in_chans,
    classes=CFG.target_size,
    activation=None,
)
```

lr = 6e-5
optimizer=torch.optim.AdamW
torch.optim.lr_scheduler.OneCycleLR

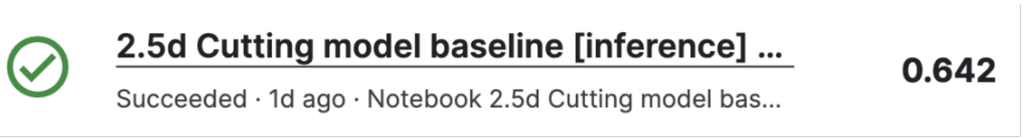
backbone	weight	loss_fn	image_size	epoch	leaderboard
se_resnext50_32x4d (Params : 25M)	imagenet	FocalLoss	256	15	0.252
se_resnext50_32x4d (Params : 25M)	imagenet	BCEWithLogitsLoss	256	15	0.431
se_resnext50_32x4d (Params : 25M)	imagenet	DiceLoss	256	15	0.556
densenet161 (Params : 26M)	instagram	DiceLoss	256	15	0.577
efficientnet-b6 (Params : 40M)	imagenet	DiceLoss	256	15	0.564

(MODELING)

U-Net

```
self.encoder = smp.Unet(
    encoder_name=CFG.backbone,
    encoder_weights=weight,
    in_channels=CFG.in_chans,
    classes=CFG.target_size,
    activation=None,
)
```

lr = 6e-5
optimizer=torch.optim.AdamW
torch.optim.lr_scheduler.OneCycleLR



backbone	weight	loss_fn	image_size	epoch	leaderboard
se_resnext101 _32x4d (Params : 46M)	imagenet	DiceLoss	256	30	0.596
resnext101 _32x8d (Params : 86M)	imagenet	DiceLoss	256	30	0.642
resnext101 _32x16d (Params : 191M)	instagram	DiceLoss	256	40	0.582
resnext101 _32x8d (Params : 86M)	imagenet	DiceLoss	512	40	0.617

(MODELING)

3D(slice)

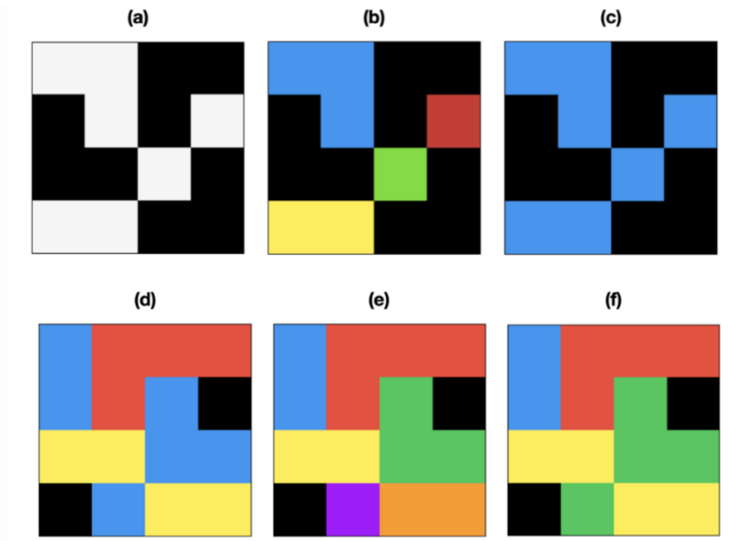


Fig. 1. Binary and Multilabel Connected Components Labeling (CCL) 2D images are shown for simplicity. (a) A binary image (foreground white, background black) (b) 4-connected CCL of binary image (c) 8-connected CCL of binary image (d) A multilabel image (e) 4-connected CCL of multilabel image (f) 8-connected CCL of multilabel image

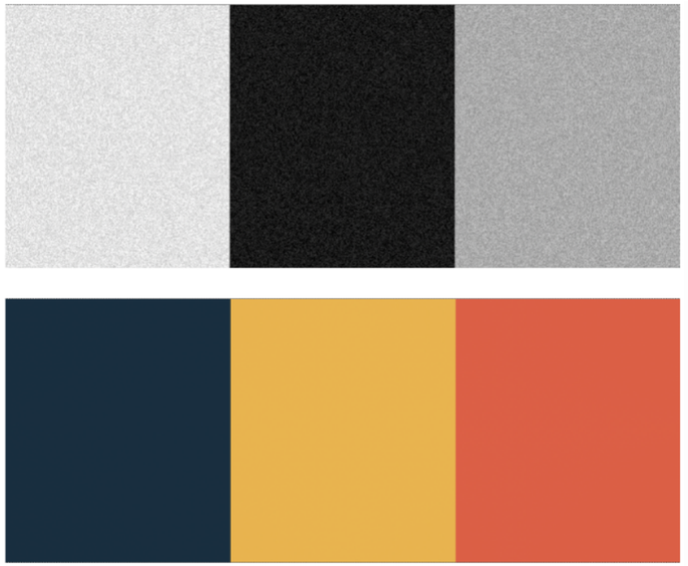


Fig. 2. Continuous Value Connected Components Labeling (CCL) (top) A three tone grayscale image with signed additive low magnitude noise (bottom) Extracted components using continuous value CCL with a delta value greater than the noise magnitude but smaller than the difference between tones

(MODELING)

3D(slice)

connected-components-3d

- 3차원 이미지에서 연결된 요소(connected components)를 식별하고 레이블링하는 데 사용되는 라이브러리.
- 연결된 요소 분석은 각각의 독립적인 객체를 식별하고 분리하는 과정.

1. 3차원 연결성 분석
: 3차원 그리드에서 서로 연결된 픽셀/복셀(3차원 픽셀) 그룹을 식별한다.

2. Labeling
: 각각의 연결된 요소에 고유한 레이블을 할당하여 다른 구성 요소와 구별함으로써, 각 구성 요소를 개별적으로 처리할 수 있다.

이를 통해 3D 이미지 데이터에서 segmentation을 수행하는 데 최적화된 기능을 사용할 수 있다.

(MODELING)

3D(Slice)

```python
for t in range(num_valid):

    if axis == 0:
        image = volume[loader[t].tolist()]
    if axis == 1:
        image = volume[:, loader[t].tolist()]
        image = image.transpose(1, 0, 2)
    if axis == 2:
        image = volume[:, :, loader[t].tolist()]
        image = image.transpose(2, 0, 1)

    batch_size, bh, bw = image.shape
    m = image.reshape(batch_size, -1)
    m = (m - m.min(keepdims=True)) / (m.max(keepdims=True) - m.min(keepdims=True) + 0.001)
    m = m.reshape(batch_size, bh, bw)
    m = np.ascontiguousarray(m)
    image = torch.from_numpy(m).float().cuda().unsqueeze(1)
```

```python
with torch.cuda.amp.autocast(enabled=True):
    with torch.no_grad():
        v, k = net(image)
        vessel += v
        kidney += k
        counter += 1

        v, k = net(torch.flip(image, dims=[2,]))
        vessel += torch.flip(v, dims=[2,])
        kidney += torch.flip(k, dims=[2,])
        counter += 1

        v, k = net(torch.flip(image, dims=[3,]))
        vessel += torch.flip(v, dims=[3,])
        kidney += torch.flip(k, dims=[3,])
        counter += 1

        v, k = net(torch.rot90(image, k=1, dims=[2,3]))
        vessel += torch.rot90(v, k=-1, dims=[2,3])
        kidney += torch.rot90(k, k=-1, dims=[2,3])
        counter += 1

        v, k = net(torch.rot90(image, k=2, dims=[2,3]))
        vessel += torch.rot90(v, k=-2, dims=[2,3])
        kidney += torch.rot90(k, k=-2, dims=[2,3])
        counter += 1

        v, k = net(torch.rot90(image, k=3, dims=[2,3]))
        vessel += torch.rot90(v, k=-3, dims=[2,3])
        kidney += torch.rot90(k, k=-3, dims=[2,3])
        counter += 1
```

(MODELING)

3D(Slice)

1. transpose 함수
: 3D 이미지 데이터를 다른 축을 기준으로 슬라이스하여 3차원 볼륨을 다양한 방향에서 보기 위함.

2. inference only
:추론 시에 메모리 사용량을 줄이기 위해 gradient 계산 X.

3. pre-trained model
: 학습된 모델(UNet)을 사용하여 각 슬라이스에 대한 신장(kidney)과 혈관(vessel)의 존재를 예측.

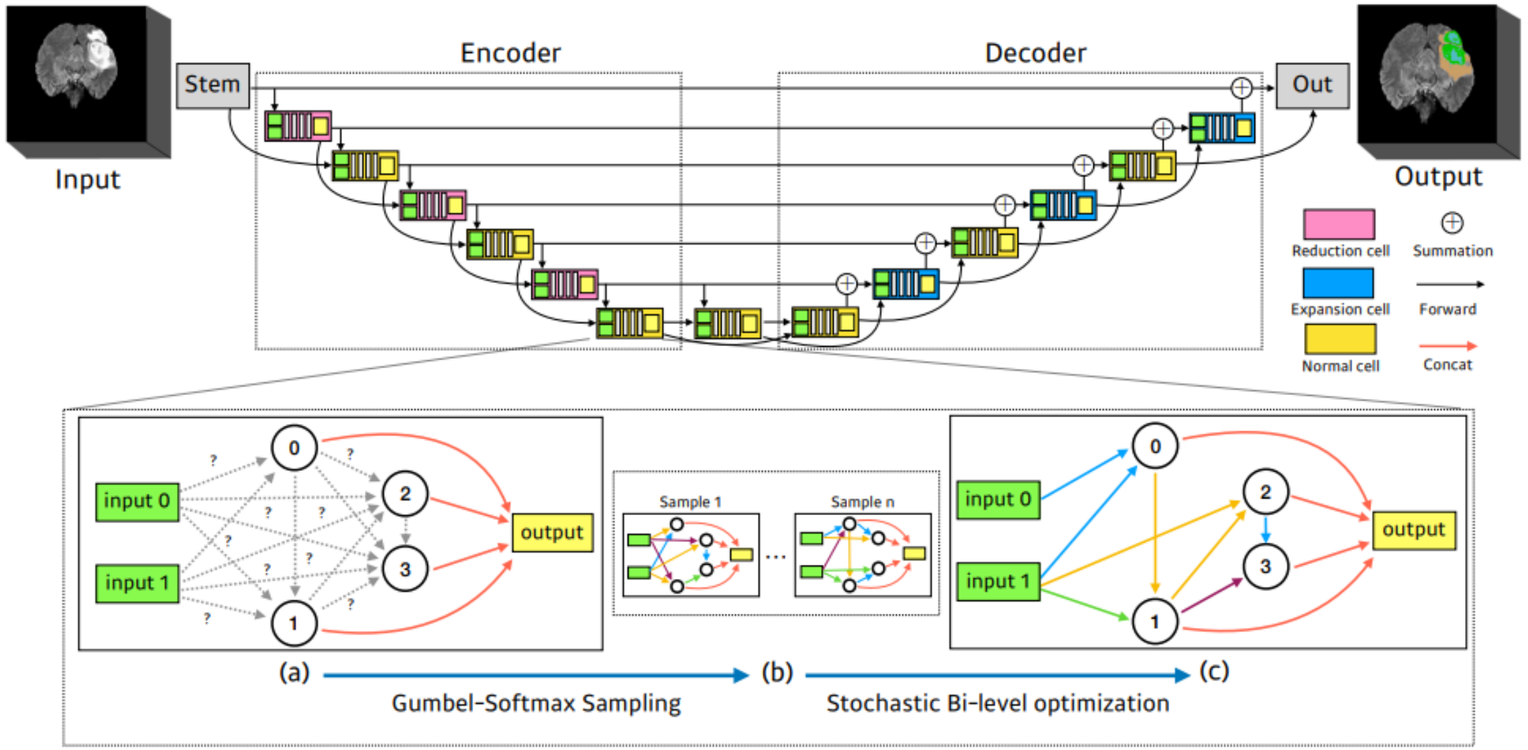
4. data augmentation
: 이미지를 뒤집거나 회전시켜 다양한 방향에서의 예측을 수행하여 xy, yz, zx의 다양한 시점에서의 정확도 향상.

(MODELING)

SCNAS

3D model

Neural Architecture Search (NAS) framework proposed for 3D medical image segmentation



(MODELING)

SCNAS

3D model

Batch size = 4,
subvolume size = 128, 128, 128
각 subvolume으로 나눠서 모델 학습 및 inference

```
next(iter(loader))[0].shape
```

✓ 0.0s

```
torch.Size([4, 1, 128, 128, 128])
```

```python
def __getitem__(self, idx):
    """
    Retrieve a subvolume by index.
    """
    z = (idx // (self.num_subvols_y * self.num_subvols_x)) * self.stride
    y = ((idx % (self.num_subvols_y * self.num_subvols_x)) // self.num_subvols_x) * self.stride
    x = ((idx % (self.num_subvols_y * self.num_subvols_x)) % self.num_subvols_x) * self.stride

    subvolume = self.volume[:, z:z + self.subvol_size, y:y + self.subvol_size, x:x + self.subvol_size]

    return torch.from_numpy((subvolume / 65536).astype(np.float32)), (z, y, x) # Convert to PyTorch tensor
```

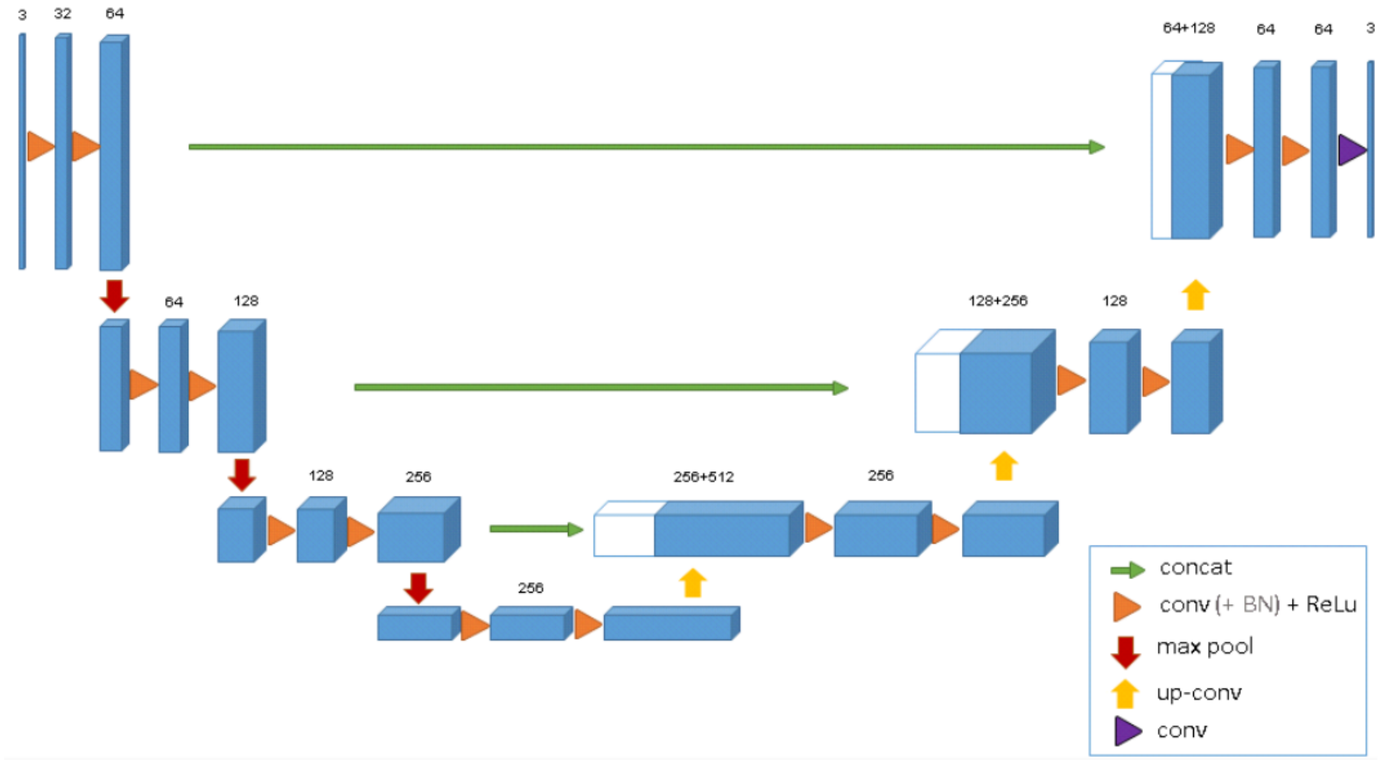
(MODELING)

3D U-Net

3D model

같은 방식을 사용하여 3D UNet
모델도 학습 및 inference 진행

그러나 3D Volume을 사용한 모델들의
성능은 좋지 않았음.



(CONCLUSION)

결론

- 3D 기반 이미지 데이터는 2D 모델의 학습만으로는 한계가 존재함 (ex. SAM).
- Training 과정도 중요하지만 제출 형식에 따른 inference 과정 역시 최종 점수 향상에 큰 영향을 미침.
- 최종적으로 2.5D로 학습한 모델을 활용하여 3 방향 슬라이스 이미지를 3D로 합쳤을 때,
  가장 높은 점수를 보임 (0.782).
- 2.5D 학습 코드에서 더 큰 최신 모델을 활용하여 inference 코드와 합친다면 더 좋은 점수가 기대됨.

Thank you