
# HuBMAP + HPA – Hacking the Human Body

## Progress Meeting 1 Group A

https://www.kaggle.com/competitions/hubmap-organ-segmentation

# ATM22

## Airway Tree Modeling Challenge 2022

**Pulmonary Airway Segmentation for bronchoscopic-assisted surgery navigation**

CT datasets

still waiting for the datasets...

# CONTENTS

# DATASET OVERVIEW

**Training data**
351 images, tiff format.
351 masks, rle format.

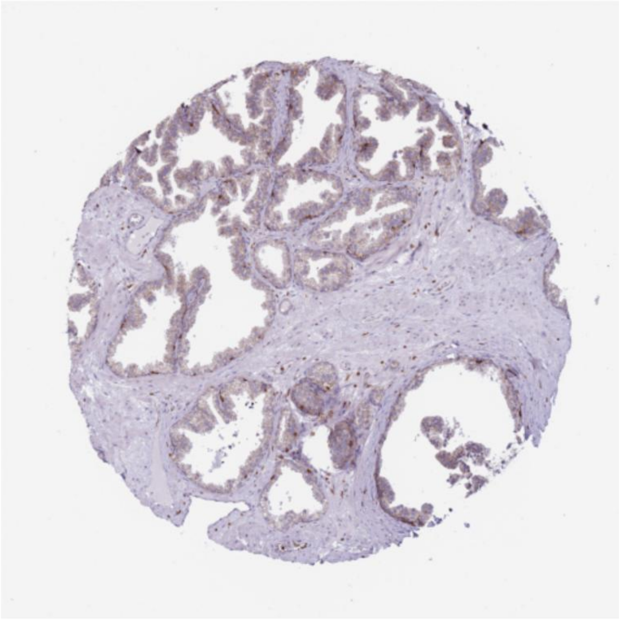
**Testing data**
1 tiff image available now.

**Mask to RLE & RLE to Mask**

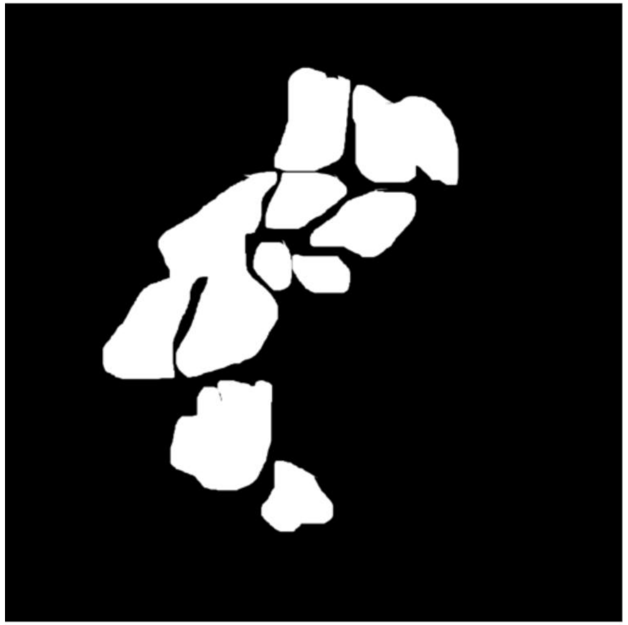
```
def mask2rle(img):
    pixels = img.T.flatten()
    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)


def rle2mask(mask_rle, shape=(1600, 256)):
    s = mask_rle.split()
    starts, lengths = [np.asarray(x, dtype=int) for x in (s[0:][::2], s[1:][::2])]
    starts -= 1
    ends = starts + lengths
    img = np.zeros(shape[0] * shape[1], dtype=np.uint8)
    for lo, hi in zip(starts, ends):
        img[lo:hi] = 1
    return img.reshape(shape).T
```

*Reference: https://www.kaggle.com/paulorzp/rle-functions-run-length-encode-decode*

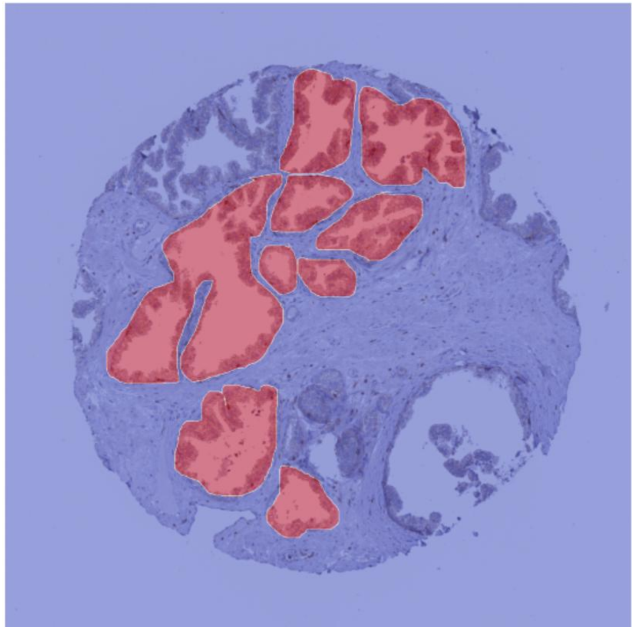
# DATASET OVERVIEW



Image



Mask



Merge

# DATASET OVERVIEW

## Data are not the same size

```
print(pd.value_counts(train_df['img_height']))
```

```
3000    326
2631      2
2416      2
2942      2
2790      2
2764      2
2654      2
2539      1
2680      1
2727      1
2308      1
2867      1
2783      1
2869      1
2760      1
2630      1
2511      1
2593      1
2675      1
3070      1
Name: img_height, dtype: int64
```

Each picture has the same height and width, and there are three channels.

However, the shape of the data are not the same.

Most of data are 3000 x 3000 x 3, a few of them are less or lager than 3000 x 3000 x 3.

```
test_image.shape
```

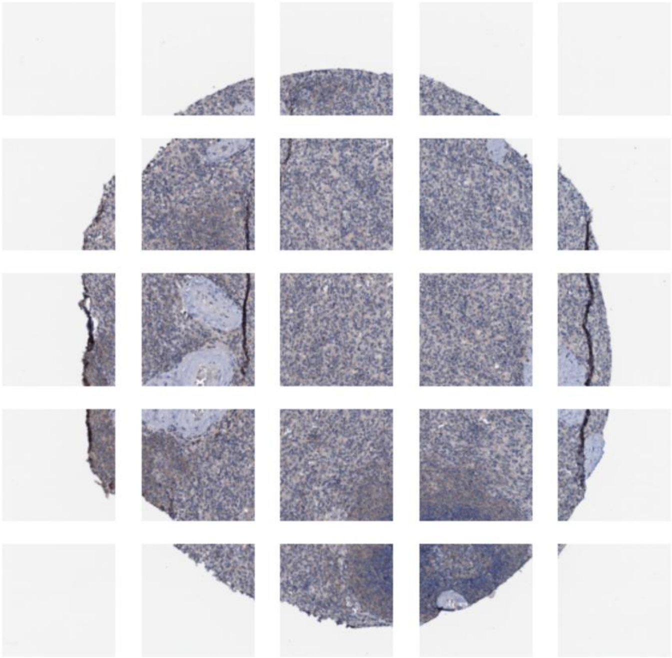
```
(2023, 2023, 3)
```

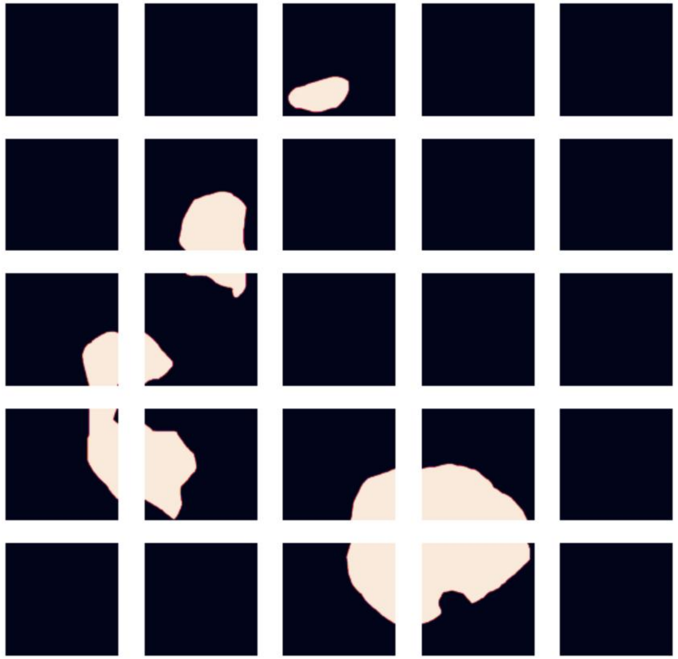
Testing data is 2023 x 2023 x 3.

# PATCHIFY DATA

**Strategy**

Because the size of the data is (3000, 3000, 3), such a large size can easily lead to CUDA out of memory. Thus, I am going to divided the data into (600, 600, 3). The reason why I choose 600 is that 3000 can be divided equally by 600 without overlapping.



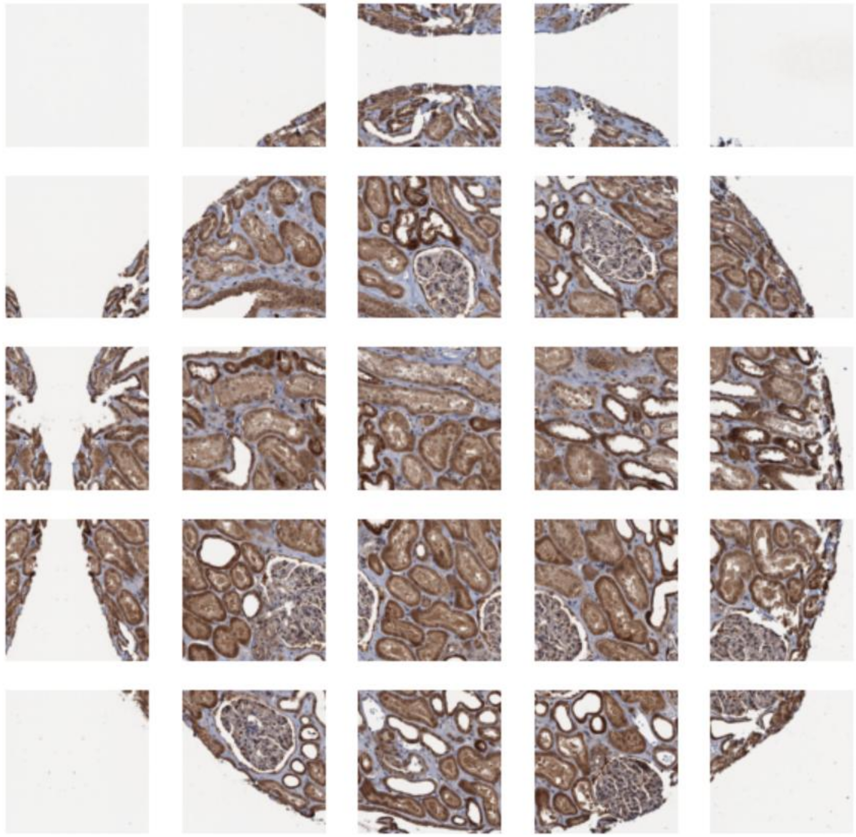


One (3000, 3000, 3) size data can be divided into twenty-five (600, 600, 3) size patches.

# PATCHIFY DATA

Data less than 3000 will be padding to 3000, than divided into patches with shape (600, 600, 3).

Data more than 3000 will be cropped to 3000 , than divided into patches with shape (600, 600, 3).



Top and left of this image has been padding

(2631, 2631, 3)
->
(3000, 3000, 3)
->
(5, 5, 600, 600, 3)

# 1. Unet

```python
def build_model():
    model = smp.Unet(
        encoder_name=CFG.backbone,
        encoder_weights=None,
        in_channels=3,
        classes=CFG.num_classes,
        activation=None,
    )
    model.to(CFG.device)
    return model

def load_model(path):
    model = build_model()
    model.load_state_dict(torch.load(path))
    model.eval()
    return model
```

```python
class CFG:
    seed = 0
    batch_size = 16
    head = "UNet"
    backbone = "efficientnet-b0"
    img_size = [512, 512]
    lr = 1e-3
    scheduler = 'CosineAnnealingLR' #['CosineAnnealingLR']
    epochs = 20
    warmup_epochs = 2
    n_folds = 5
    folds_to_run = [0]
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    base_path = '../input/hubmap-organ-segmentation'
    num_workers = mp.cpu_count()
    num_classes = 1
    n_accumulate = max(1, 16//batch_size)
    loss = 'Dice'
    optimizer = 'Adam'
    weight_decay = 1e-6
    ckpt_path = '../input/hubmap-unet-semantic-approach-train/last_epoch-00.bin' #Checkpoin
t path
    threshold = 0.5
```

| | |
|---|---|
| Succeeded | 0.28 |

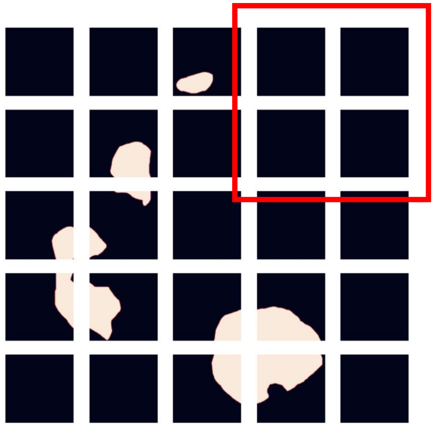
```
964/987, train_loss: 0.7337
965/987, train_loss: 0.7013
966/987, train_loss: 0.8712
967/987, train_loss: 0.6988
968/987, train_loss: 0.8794
969/987, train_loss: 0.7476
970/987, train_loss: 0.8425
971/987, train_loss: 0.7392
972/987, train_loss: 0.8207
973/987, train_loss: 0.8207
974/987, train_loss: 0.8441
975/987, train_loss: 0.8524
976/987, train_loss: 0.8935
977/987, train_loss: 0.8761
978/987, train_loss: 0.7955
979/987, train_loss: 0.9990
980/987, train_loss: 0.8294
981/987, train_loss: 0.5218
982/987, train_loss: 0.7062
983/987, train_loss: 0.8572
984/987, train_loss: 0.7529
985/987, train_loss: 0.7659
986/987, train_loss: 0.8410
987/987, train_loss: 0.8420
988/987, train_loss: 1.0000
epoch 2 average loss: 0.7992
saved new best metric model
current epoch: 2 current mean dice: 0.3497 best mean dice: 0.3497 at epoch 2
train completed, best_metric: 0.3497 at epoch: 2
```

## 03 DeeplabV3+

```
epoch:  13 loss:   0.004 accuracy: 0.986 IOU: 0.838 test_loss:   0.023 test_accuracy: 0.952 test_iou: 0.522

100%|██████████| 988/988 [04:16<00:00,  3.86it/s]
100%|██████████| 110/110 [00:14<00:00,  7.70it/s]

epoch:  14 loss:   0.004 accuracy: 0.987 IOU: nan test_loss:   0.023 test_accuracy: 0.957 test_iou: 0.548

100%|██████████| 988/988 [04:21<00:00,  3.78it/s]
100%|██████████| 110/110 [00:14<00:00,  7.83it/s]

epoch:  15 loss:   0.004 accuracy: 0.989 IOU: nan test_loss:   0.02 test_accuracy: 0.961 test_iou: 0.575

100%|██████████| 988/988 [04:32<00:00,  3.63it/s]
100%|██████████| 110/110 [00:19<00:00,  5.70it/s]

epoch:  16 loss:   0.004 accuracy: 0.988 IOU: nan test_loss:   0.043 test_accuracy: 0.931 test_iou: 0.349

100%|██████████| 988/988 [04:49<00:00,  3.41it/s]
100%|██████████| 110/110 [00:18<00:00,  5.85it/s]

epoch:  17 loss:   0.004 accuracy: 0.987 IOU: nan test_loss:   0.018 test_accuracy: 0.964 test_iou: 0.627


100%|██████████| 988/988 [04:31<00:00,  3.64it/s]
100%|██████████| 110/110 [00:13<00:00,  7.96it/s]

epoch:  18 loss:   0.003 accuracy: 0.991 IOU: nan test_loss:   0.019 test_accuracy: 0.966 test_iou: 0.638

100%|██████████| 988/988 [04:18<00:00,  3.82it/s]
100%|██████████| 110/110 [00:18<00:00,  5.88it/s]

epoch:  19 loss:   0.003 accuracy: 0.99 IOU: 0.873 test_loss:   0.026 test_accuracy: 0.951 test_iou: 0.484

100%|██████████| 988/988 [04:31<00:00,  3.64it/s]
100%|██████████| 110/110 [00:14<00:00,  7.57it/s]

epoch:  20 loss:   0.004 accuracy: 0.988 IOU: nan test_loss:   0.018 test_accuracy: 0.96 test_iou: nan
```

# 04 Further improvement

**1. Skip masks with all zeros**



**2. Data Augmentation**

Data augmentation is useful to improve performance and outcomes of machine learning models by forming new and different examples to train datasets. If dataset in a machine learning model is rich and sufficient, the model performs better and more accurate.
https://www.kaggle.com/code/thedevastator/strip-ai-image-augmentations-tutorial

**Classical Methods**

**Classic image processing activities for data augmentation are:**

- Padding
- Random rotating
- Re-scaling,
- Vertical and horizontal flipping
- Translation ( image is moved along X, Y direction)
- Cropping
- Zooming
- Darkening & brightening/color modification
- Grayscaling
- Changing contrast
- Adding noise
- Random erasing

**3. Pretrain a weight using other datasets**