2. Solution

(1) 1st Solution Summary

Private Score 0.9515 / Public Score 0.9360

[Summary]

- single model **Unet se_resnext101_32x4d** (4folds)
- **some techniques** from previous segmentation competitions (mainly from the cloud comp.)
- **balanced tile sampling** for training (EDIT : masked area is balanced)
- **pseudo-label** for the public test data and external data
- a trick for **avoiding** the **edge effect**
- My pipeline was developed with the old dataset (i.e., before the data update).

2. Solution

(1) 1st Solution Summary

Private Score 0.9515 / Public Score 0.9360

[Data Preparation]

- data_preparation_01_01~02.py
- make 1024x1024 tiles + shifted 1024x1024 tiles (I shifted the tiles by (512,512))

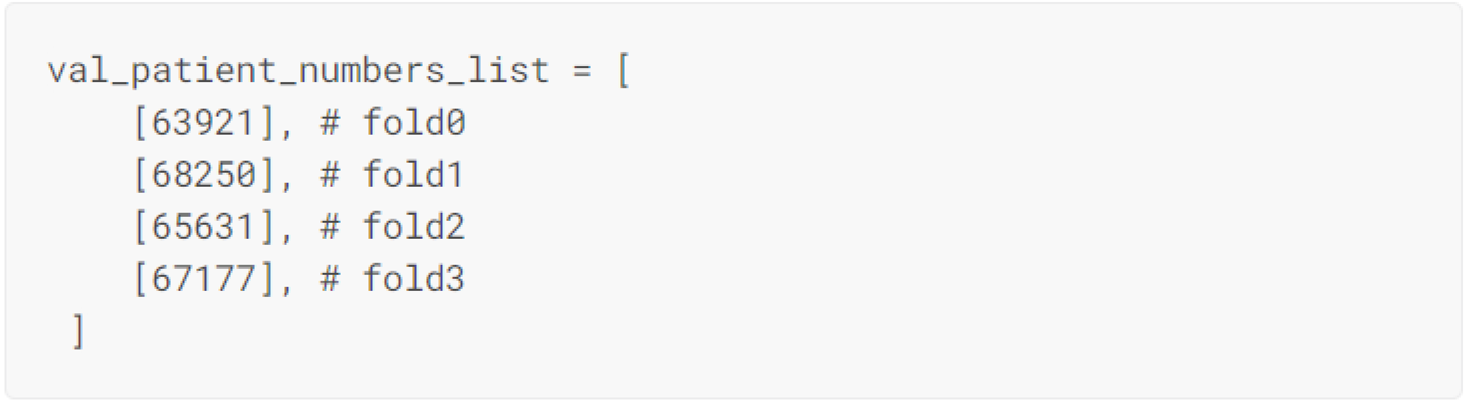
2. Solution

(1) 1st Solution Summary

Private Score 0.9515 / Public Score 0.9360

[Validation]

- I selected the validation data so that the same patient number is in the same group

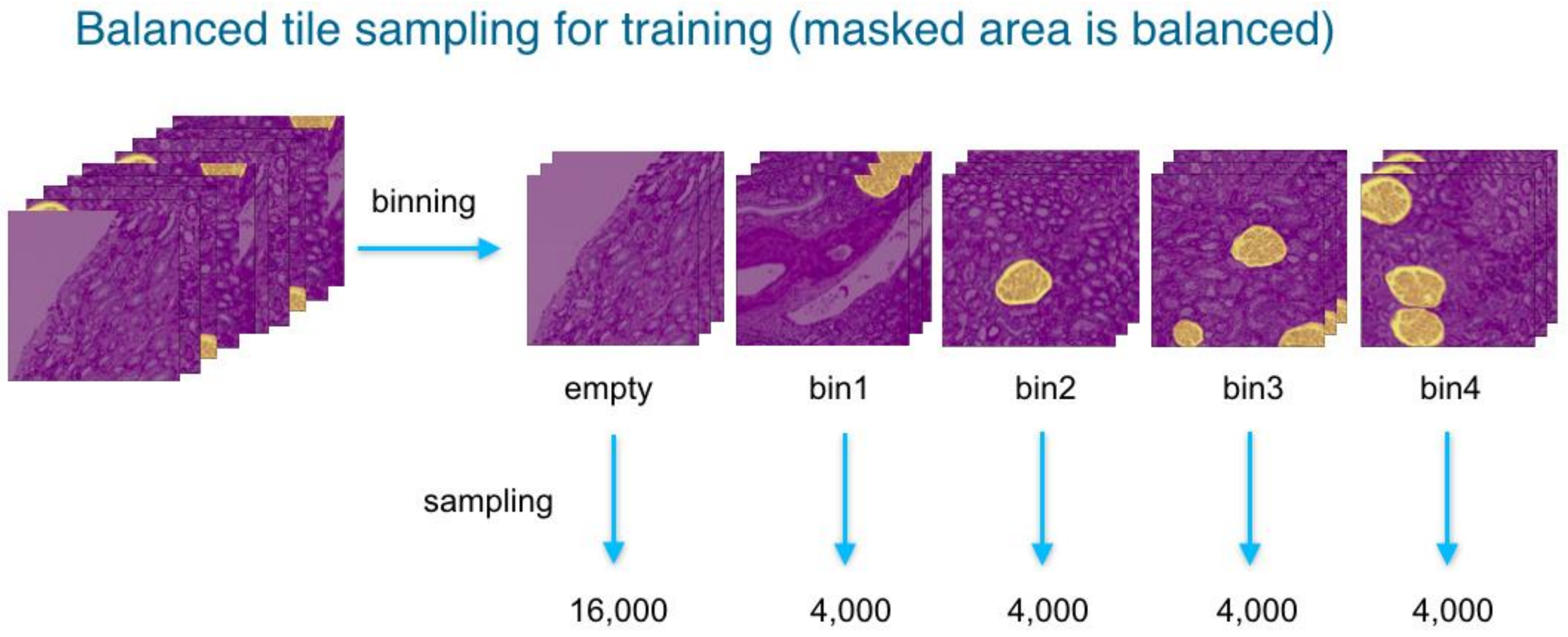


```
val_patient_numbers_list = [
    [63921], # fold0
    [68250], # fold1
    [65631], # fold2
    [67177], # fold3
 ]
```

2. Solution
(1) 1st Solution Summary

Private Score 0.9515 / Public Score 0.9360

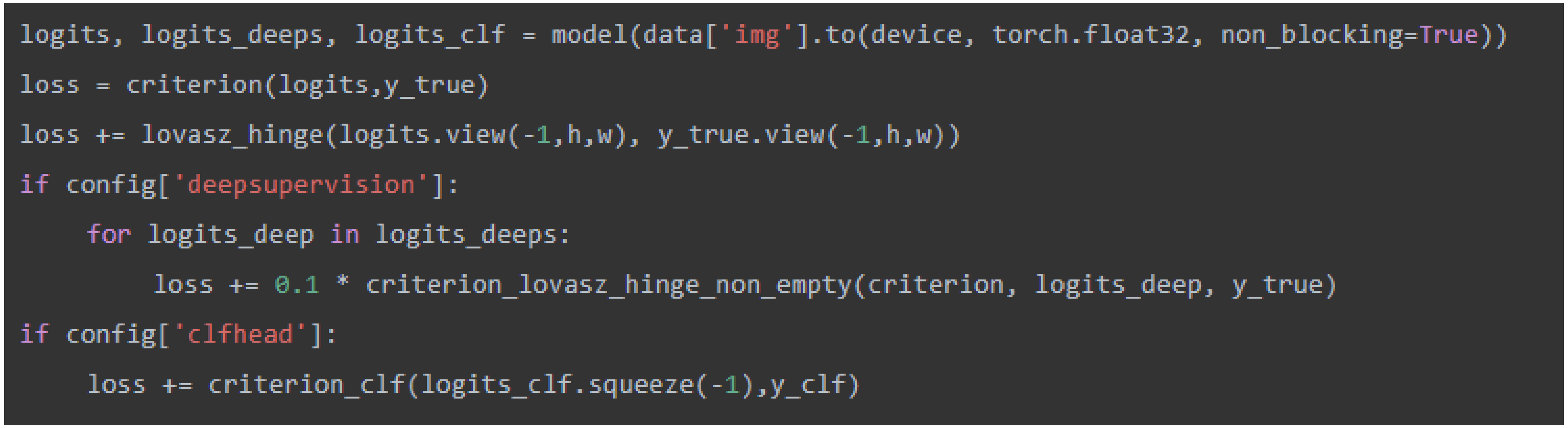
[Balanced tile sampling for training]
- First I binned the tile data with respect to the masked area (number of bins = 4 for masked tiles). Then I apply the following procedure for balanced sampling.



2. Solution
(1) 1st Solution Summary

[Loss]

- I used **bce loss + lovasz-hinge loss**, on top oh that I used **deep supervision with bce loss + lovasz-hinge loss (for only non-empty masks) multiplied by 0.1**. For classification head, I used **bce loss**



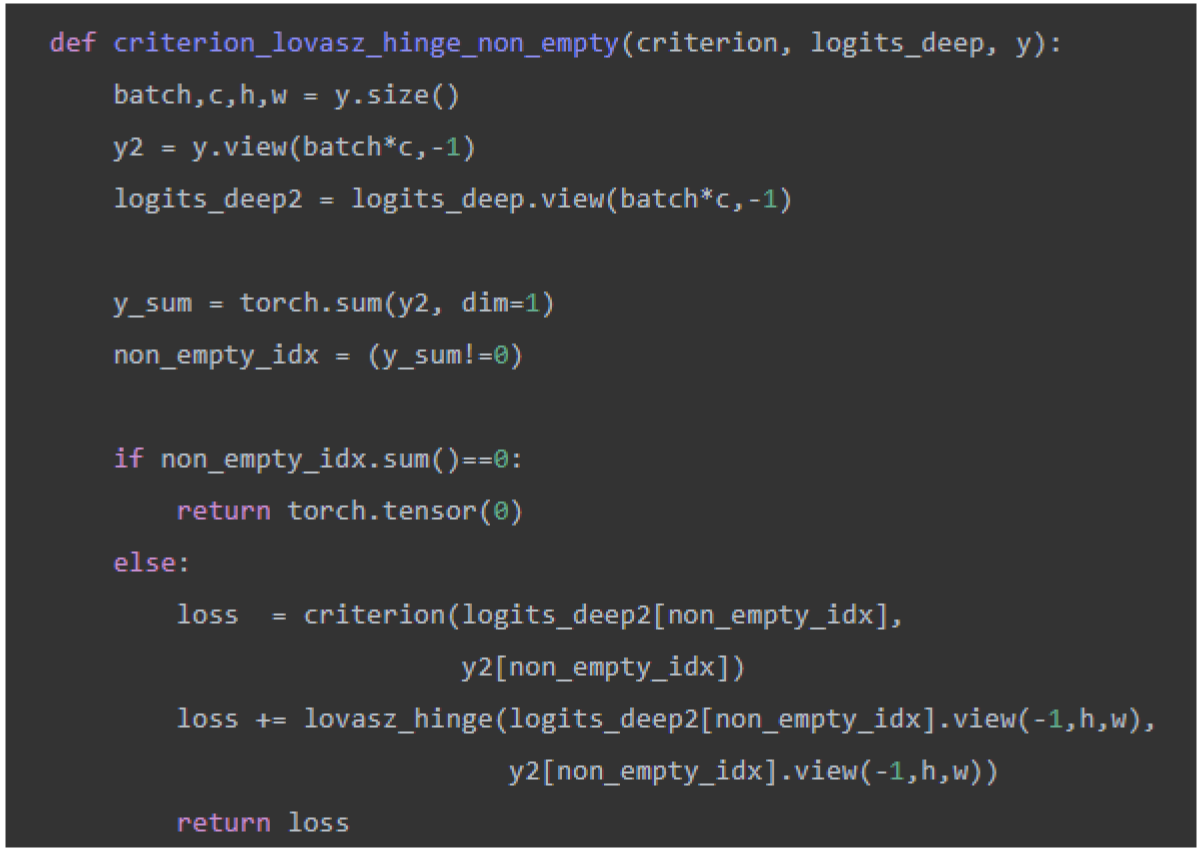
```
logits, logits_deeps, logits_clf = model(data['img'].to(device, torch.float32, non_blocking=True))
loss = criterion(logits,y_true)
loss += lovasz_hinge(logits.view(-1,h,w), y_true.view(-1,h,w))
if config['deepsupervision']:
    for logits_deep in logits_deeps:
        loss += 0.1 * criterion_lovasz_hinge_non_empty(criterion, logits_deep, y_true)
if config['clfhead']:
    loss += criterion_clf(logits_clf.squeeze(-1),y_clf)
```

2. Solution

(1) 1st Solution Summary

[Loss]

- I used **bce loss + lovasz-hinge loss**, on top oh that I used **deep supervision with bce loss + lovasz-hinge loss (for only non-empty masks) multiplied by 0.1**. For classification head, I used **bce loss**



```
def criterion_lovasz_hinge_non_empty(criterion, logits_deep, y):
    batch,c,h,w = y.size()
    y2 = y.view(batch*c,-1)
    logits_deep2 = logits_deep.view(batch*c,-1)

    y_sum = torch.sum(y2, dim=1)
    non_empty_idx = (y_sum!=0)

    if non_empty_idx.sum()==0:
        return torch.tensor(0)
    else:
        loss  = criterion(logits_deep2[non_empty_idx],
                          y2[non_empty_idx])
        loss += lovasz_hinge(logits_deep2[non_empty_idx].view(-1,h,w),
                             y2[non_empty_idx].view(-1,h,w))
        return loss
```

2. Solution

(1) 1st Solution Summary

[External data and Pseudo-label]

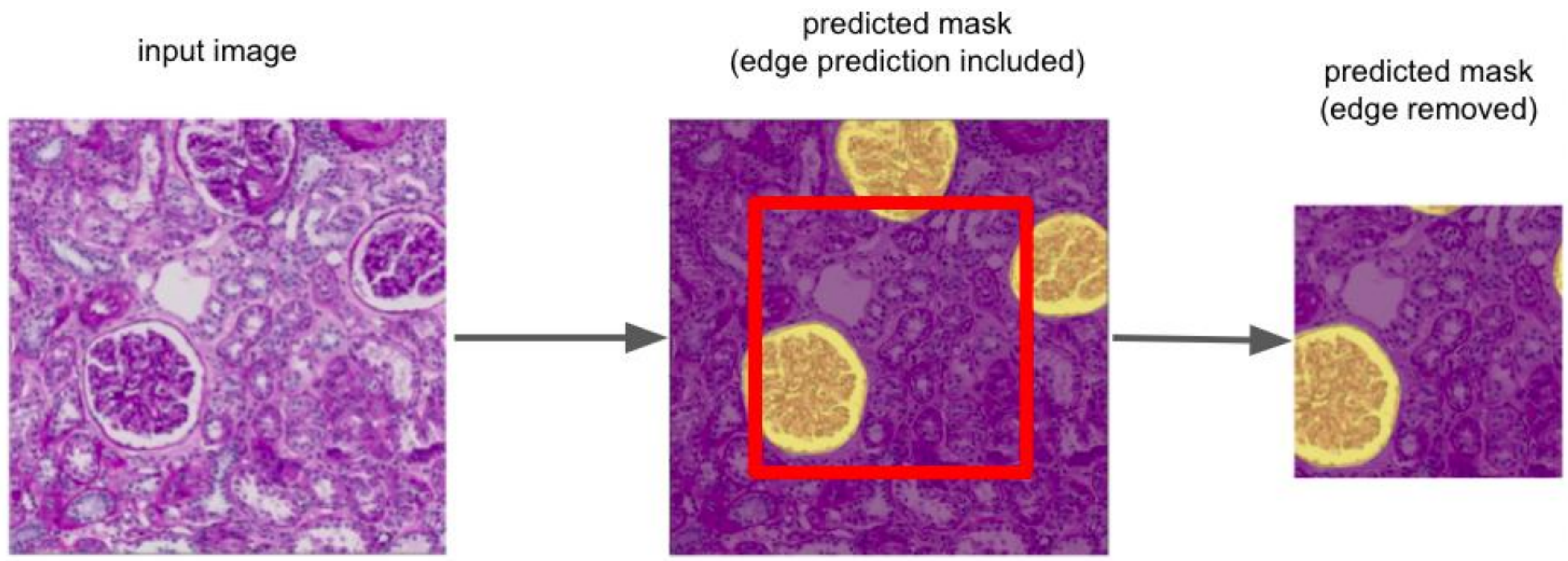
- Train dataset
- Public test dataset
- Hubmap Portal
    - https://portal.hubmapconsortium.org/search?entity_type[0]=Dataset
- Dataset_a_dib
    - https://data.mendeley.com/datasets/k7nvtgn2x6/3)

2. Solution

(1) 1st Solution Summary

[A Trick for Inference]

- Avoiding Edge Effect



2. Solution

(1) 3rd Solution Summary

- reduce=2, size=1024
- augmentation
  - cut-out
  - albumentation
- Model
  - resnext50
  - resnext101
- Inference trick
  - 오른쪽 사진
  - self.sz = 2048, self.expansion = 512, reduction = 2
  - -> 1024+256 사이즈 -> 1024로 crop

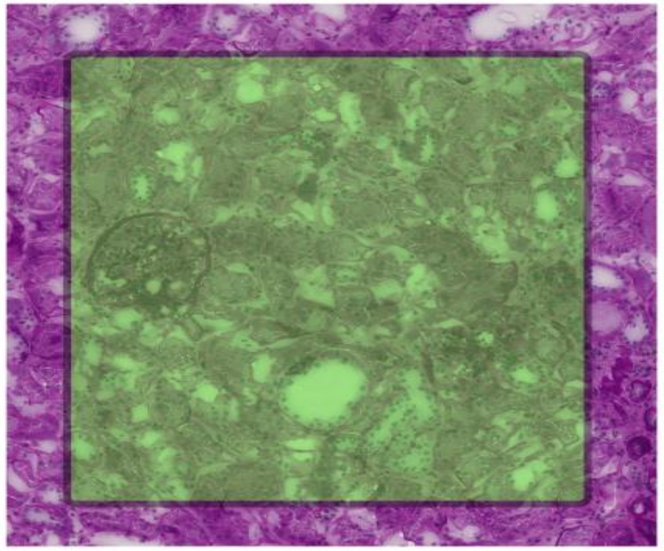
```
p00,p01 = max(0,x0), min(x0+self.sz+self.expansion,self.shape[0])
p10,p11 = max(0,y0), min(y0+self.sz+self.expansion,self.shape[1])
```

- crop :

```
yield py[i,expansion*sz_reduction//2:-expansion*sz_reduction//2,
expansion*sz_reduction//2:-expansion*sz_reduction//2],y[i]
```

```
def get_aug(p=1.0):
  return Compose([
    HorizontalFlip(),
    VerticalFlip(),
    RandomRotate90(),
    ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.2, rotate_limit=15, p=0.9,
                     border_mode=cv2.BORDER_REFLECT),
    OneOf([
      ElasticTransform(p=.3),
      GaussianBlur(p=.3),
      GaussNoise(p=.3),
      OpticalDistortion(p=0.3),
      GridDistortion(p=.1),
      IAAPiecewiseAffine(p=0.3),
    ], p=0.3),
    OneOf([
      HueSaturationValue(15,25,0),
      CLAHE(clip_limit=2),
      RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3),
    ], p=0.3),
  ], p=p)
```

Entire tile is run through the neural network,
but only green region is used as inference output

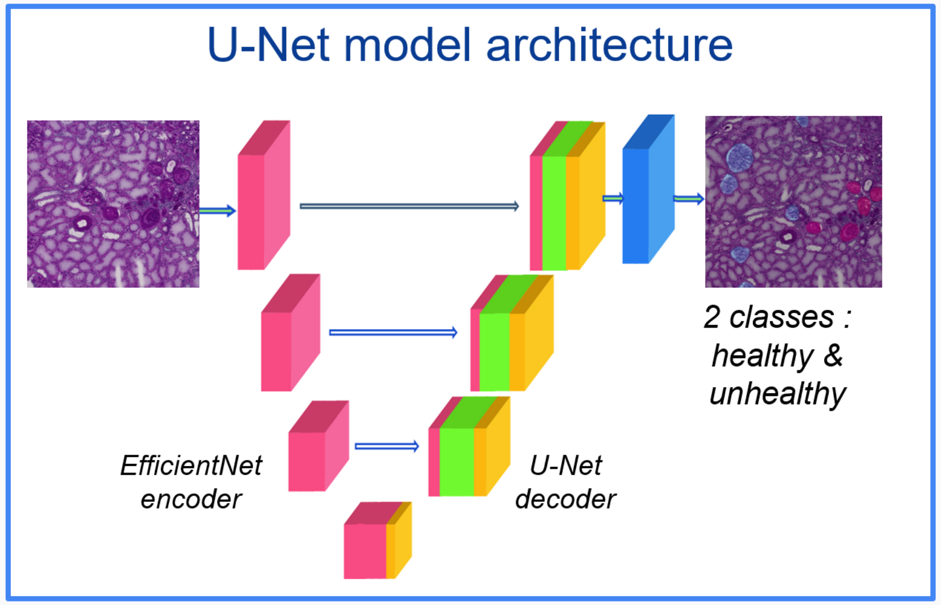


2. Solution

(1) 4th Solution Summary

[Overview]

- proper consideration of healthy and unhealthy glomeruli and trust in CV
- 2 class approach
- proper annotation of the data
- be careful with d488c759a when check LB



Generalization to new data is key

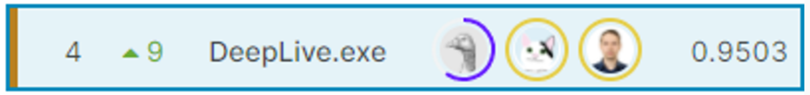
- Advanced data augmentation
- Manually labeled data from 2 external sources
- Pseudo-labeled 7 images from HuBMAP
- AI assisted reannotation of train data

Intelligent tiling for fast convergence

- Sample only interesting tiles on the fly
- Augment larger tiles to avoid edge effects

Competition results

- Validation score : **0,9439** (per image split)
- **0,940** on the public Leaderboard **(#13)**
- **0,9503** on the private one ! **(#4)**



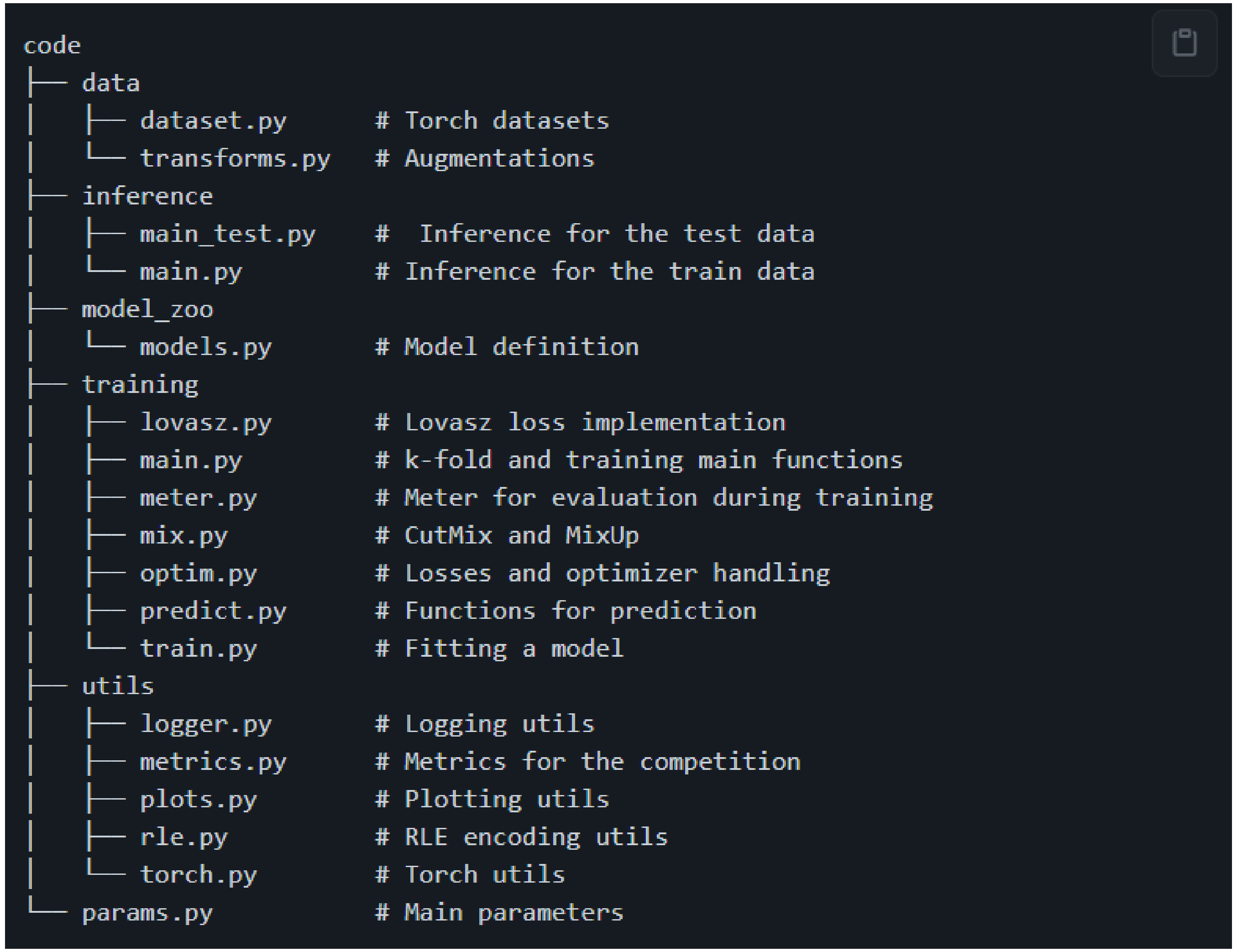
4 ▲9 DeepLive.exe 0.9503

2. Solution

(1) 4th Solution Summary

[Overview]

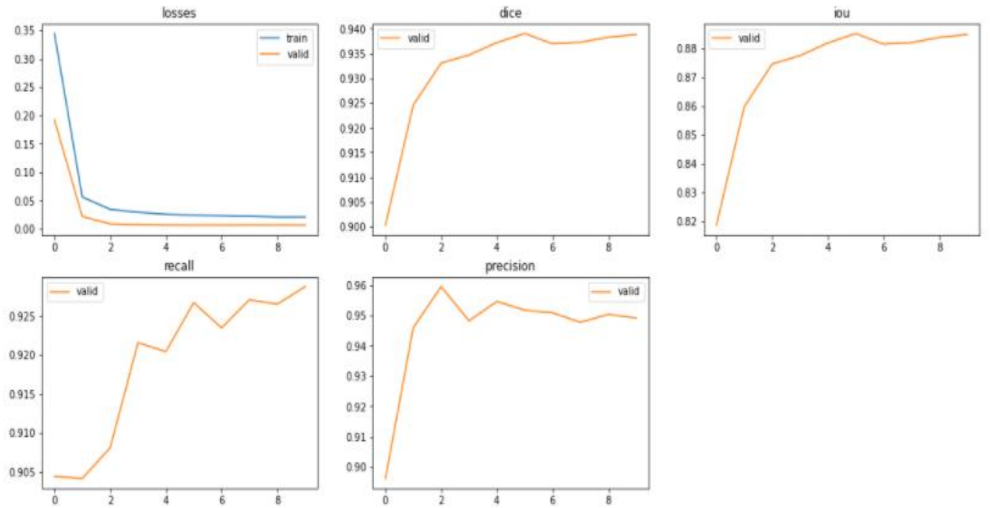
- Code Structure



```
code
├── data
│   ├── dataset.py      # Torch datasets
│   └── transforms.py   # Augmentations
├── inference
│   ├── main_test.py    #  Inference for the test data
│   └── main.py         # Inference for the train data
├── model_zoo
│   └── models.py       # Model definition
├── training
│   ├── lovasz.py       # Lovasz loss implementation
│   ├── main.py         # k-fold and training main functions
│   ├── meter.py        # Meter for evaluation during training
│   ├── mix.py          # CutMix and MixUp
│   ├── optim.py        # Losses and optimizer handling
│   ├── predict.py      # Functions for prediction
│   └── train.py        # Fitting a model
├── utils
│   ├── logger.py       # Logging utils
│   ├── metrics.py      # Metrics for the competition
│   ├── plots.py        # Plotting utils
│   ├── rle.py          # RLE encoding utils
│   └── torch.py        # Torch utils
└── params.py           # Main parameters
```

2. Solution

(1) 10th Solution Summary



- Training
  - Architecture: U-Net
  - Encoder: efficientnet-b2
  - Pretraining: imagenet
  - Loss: Dice-CrossEntropy
  - Optimizer: ranger
  - Learning rate: 1e-3
  - Batch Size: 16
  - Tile Size: 512x512
  - Resolultion Downscaling Factor: 3
  - Training iterations: 2500-3000 (best model selected on validation set)
  - Ensembling 1: 5 Models (5-fold cross validation)
  - Ensembling 2: 3 Models at scale 2,3, and 4 trained on all data
  - Augmentations: see training Notebook

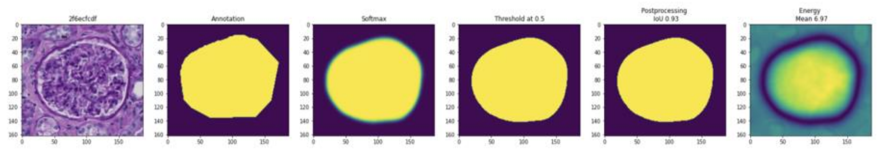


Figure 7: True positive example with high energy

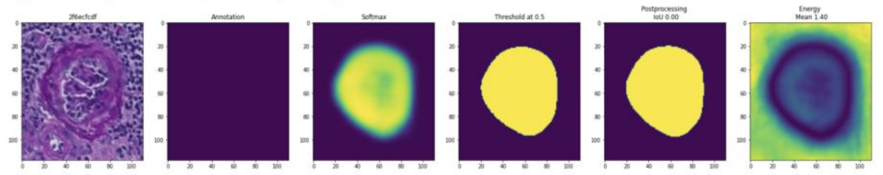


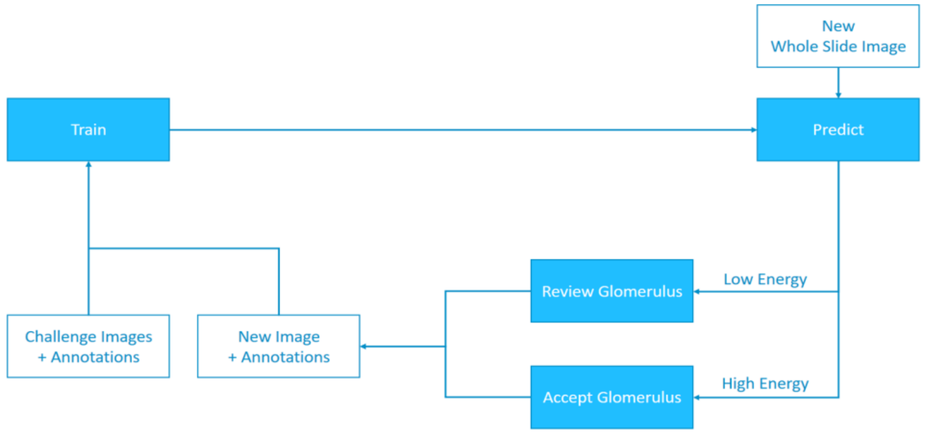
Figure 8: False positive example: using a thresholded softmax at 0.5, the network predicts a glomerulus. However, the energy is comparatively low indicating uncertainty in the prediction.

- False Positive 거르는 방법 : Energy를 이용한 confidence estimation
  - 식은 링크 참조.

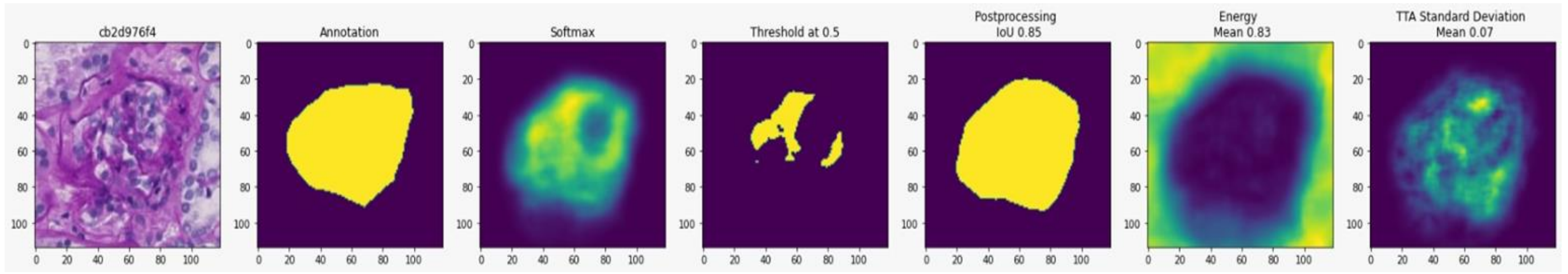
- 훈련 process
  - human-in-the-loop annotation process is ideal and helps to create more training data painlessly

2. Solution

(1) 10th Solution Summary



- Inference
  - Method
    - Overlapping tiles (shift factor 0.8)
    - Gaussian weighting (nnunet, code)
      - To suppress stitching artifacts and reduce the influence of positions close to the borders, a Gaussian importance weighting is applied, increasing the weight of the center voxels in the softmax aggregation
    - Pre-filtering of empty tiles (thanks to @iafoss (kernel)!
    - Test-time augmentation (horizontal and vertical flip)
  - post-process
    - crf는 dice score에 안좋음.
    - pixel 수가 작은걸 없애는 방법
      - 놓치는게 있음



    - Otsu's method
      - 좋으나, 이미지 전체로 하면 dice score떨어짐
      - submission에는 사용하지 않았지만,
      - validation할때 사용함.
        - 1만개 미만의 pixel로 예측된 이미지(reduction=2)에 사용

2. Solution

(1) 12th Solution Summary

[Simple Ensemble Approach]

- [1] EfficientNetB7 with 5folds -> 이 중에서 3개를 선택 (?) : **3개**
- [2] EfficientNetB7 with 5folds Other Seed -> 이 중에서 1개를 선택 (?) : **1개**
- [3] EfficientNetB5 with d488 Labels : **1개**

[Zhao's Labels]

1. [1] EfficientNetB7 Model을 이용해서 학습 : LB 0.934
2. [1] + [2]를 결합 : LB 0.935
3. 0.937에 도달할 때까지 d488을 Pesudo Labeling 작업을 진행

2. Solution

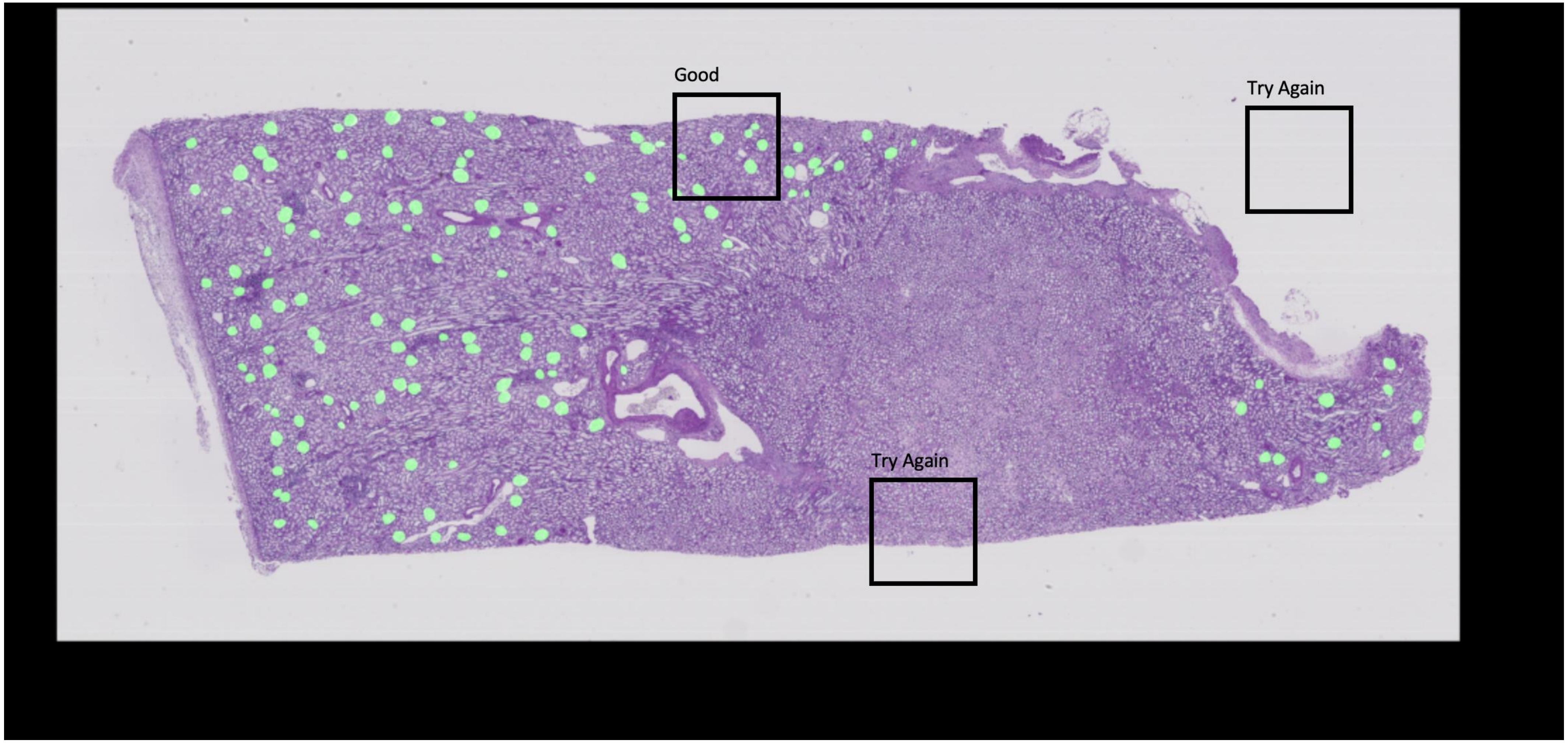
(1) 23rd Solution Summary

[Chris Solution]

- Train Data Loader

- 128개의 Good에 대해서만 학습을 진행 with 15 train images
- Numpy trick으로 크기 감소 (2, 3, 4배 감소)
  - image = image[::2, ::2,], image = image[::3, ::3,]

- 1024x1024는 적어도 1개의 segmentation label 포함

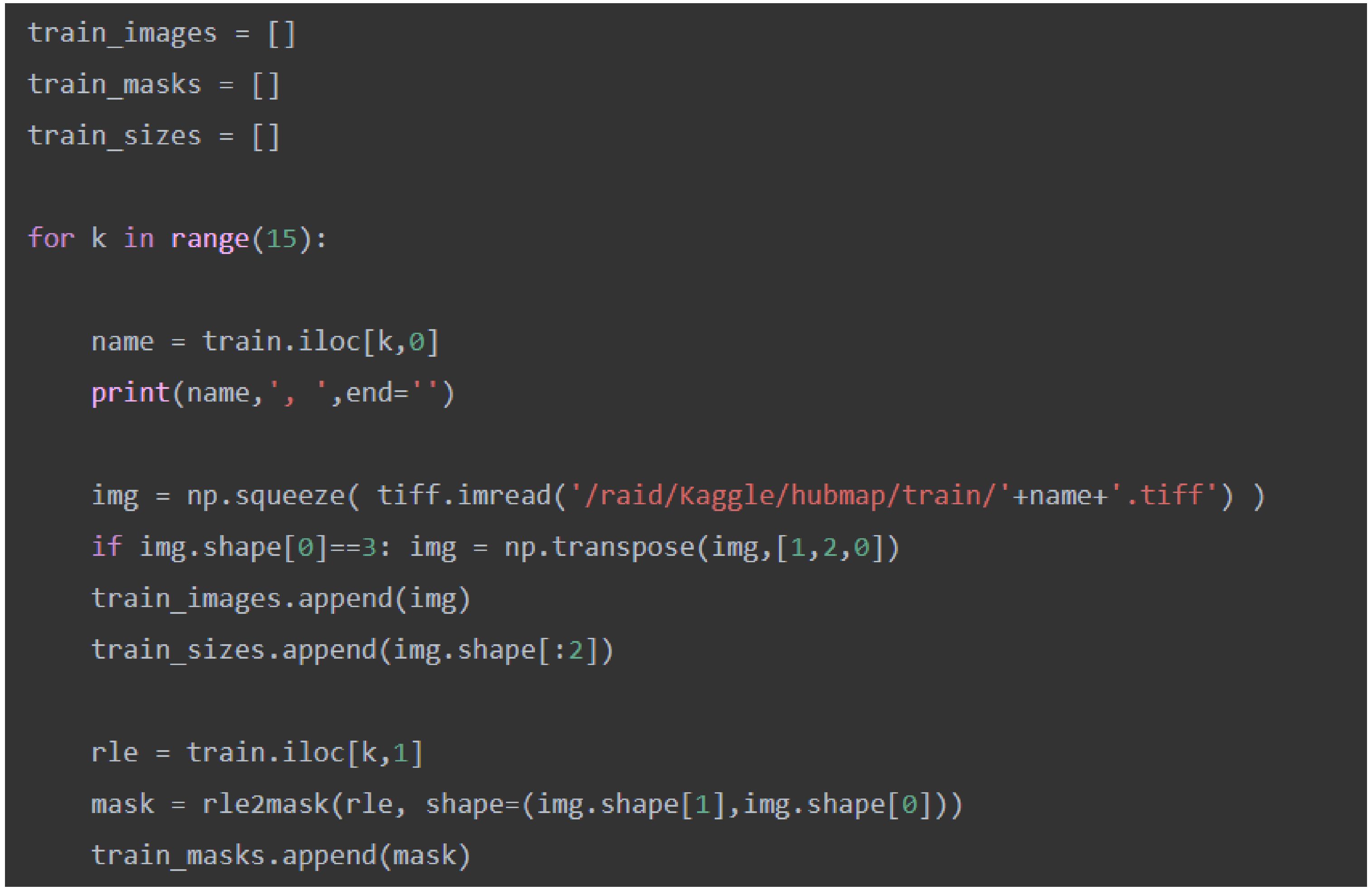


2. Solution

(1) 23rd Solution Summary

[Chris Solution]

- Train Data Loader
- 첫째, 15개의 모든 이미지를 올려놓음



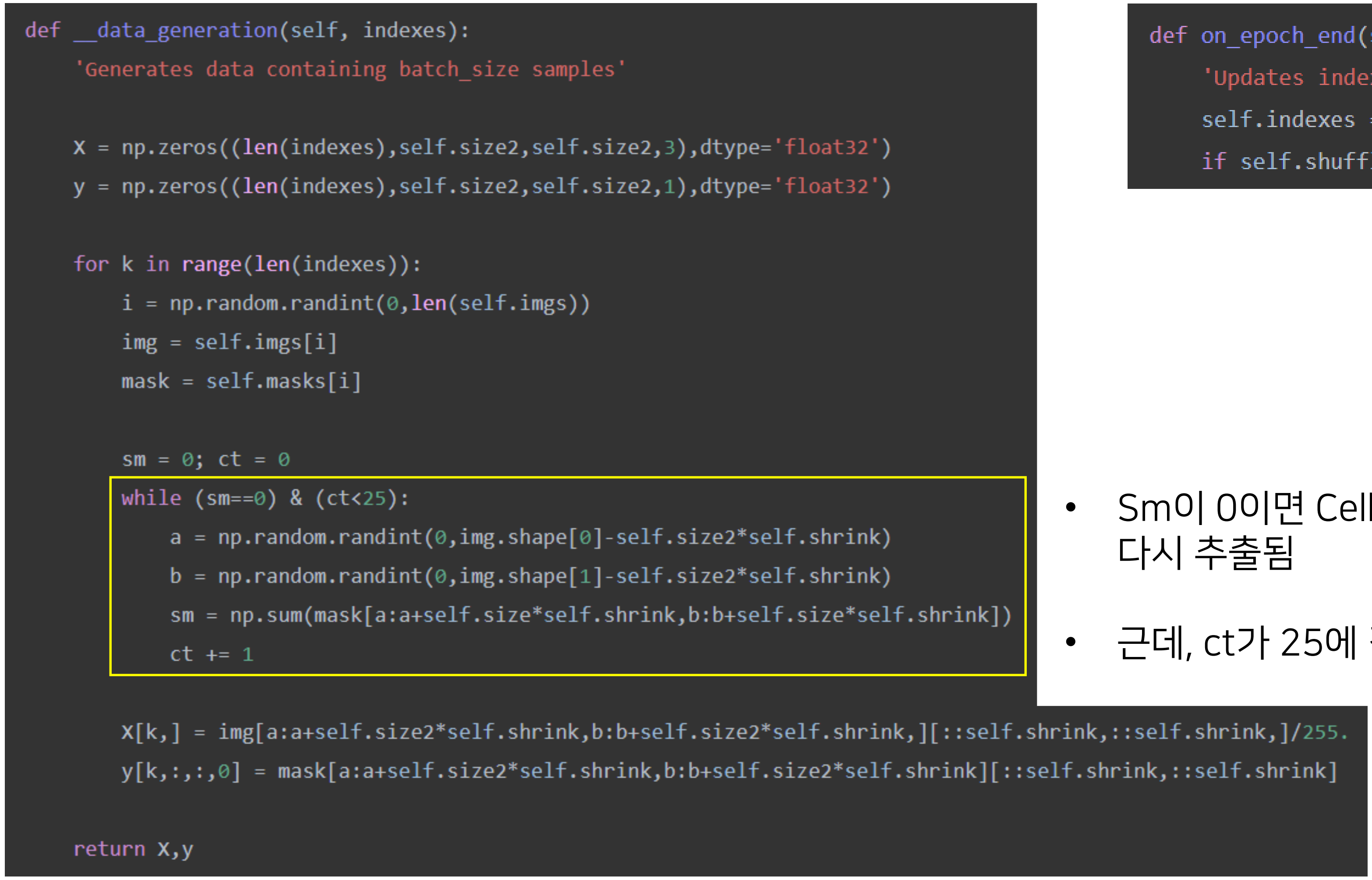
```python
train_images = []
train_masks = []
train_sizes = []

for k in range(15):

    name = train.iloc[k,0]
    print(name,', ',end='')

    img = np.squeeze( tiff.imread('/raid/Kaggle/hubmap/train/'+name+'.tiff') )
    if img.shape[0]==3: img = np.transpose(img,[1,2,0])
    train_images.append(img)
    train_sizes.append(img.shape[:2])

    rle = train.iloc[k,1]
    mask = rle2mask(rle, shape=(img.shape[1],img.shape[0]))
    train_masks.append(mask)
```

2. Solution

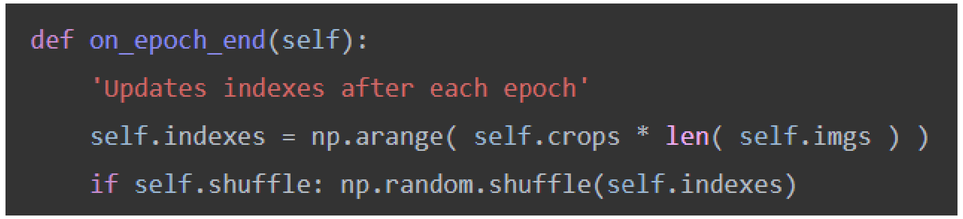
(1) 23rd Solution Summary

[Chris Solution]

- Train Data Loader
- 둘째, 반복해서 추출



```
def __data_generation(self, indexes):
    'Generates data containing batch_size samples'

    X = np.zeros((len(indexes),self.size2,self.size2,3),dtype='float32')
    y = np.zeros((len(indexes),self.size2,self.size2,1),dtype='float32')

    for k in range(len(indexes)):
        i = np.random.randint(0,len(self.imgs))
        img = self.imgs[i]
        mask = self.masks[i]

        sm = 0; ct = 0
        while (sm==0) & (ct<25):
            a = np.random.randint(0,img.shape[0]-self.size2*self.shrink)
            b = np.random.randint(0,img.shape[1]-self.size2*self.shrink)
            sm = np.sum(mask[a:a+self.size*self.shrink,b:b+self.size*self.shrink])
            ct += 1

        X[k,] = img[a:a+self.size2*self.shrink,b:b+self.size2*self.shrink,][::self.shrink,::self.shrink,]/255.
        y[k,:,:,0] = mask[a:a+self.size2*self.shrink,b:b+self.size2*self.shrink][::self.shrink,::self.shrink]

    return X,y
```



```
def on_epoch_end(self):
    'Updates indexes after each epoch'
    self.indexes = np.arange( self.crops * len( self.imgs ) )
    if self.shuffle: np.random.shuffle(self.indexes)
```

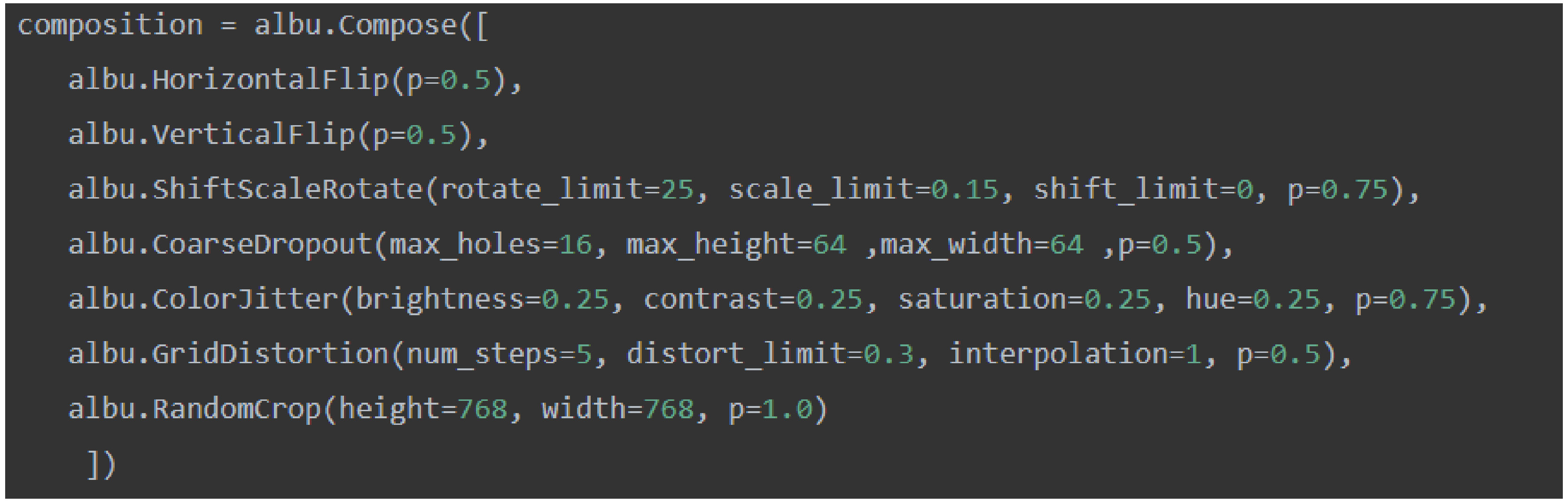
- Sm이 0이면 Cell이 아예 없다는 것으로 While이 다시 돌아가서 다시 추출됨
- 근데, ct가 25에 걸리면 아닌 부분도 학습에 참여할 수 있음

2. Solution

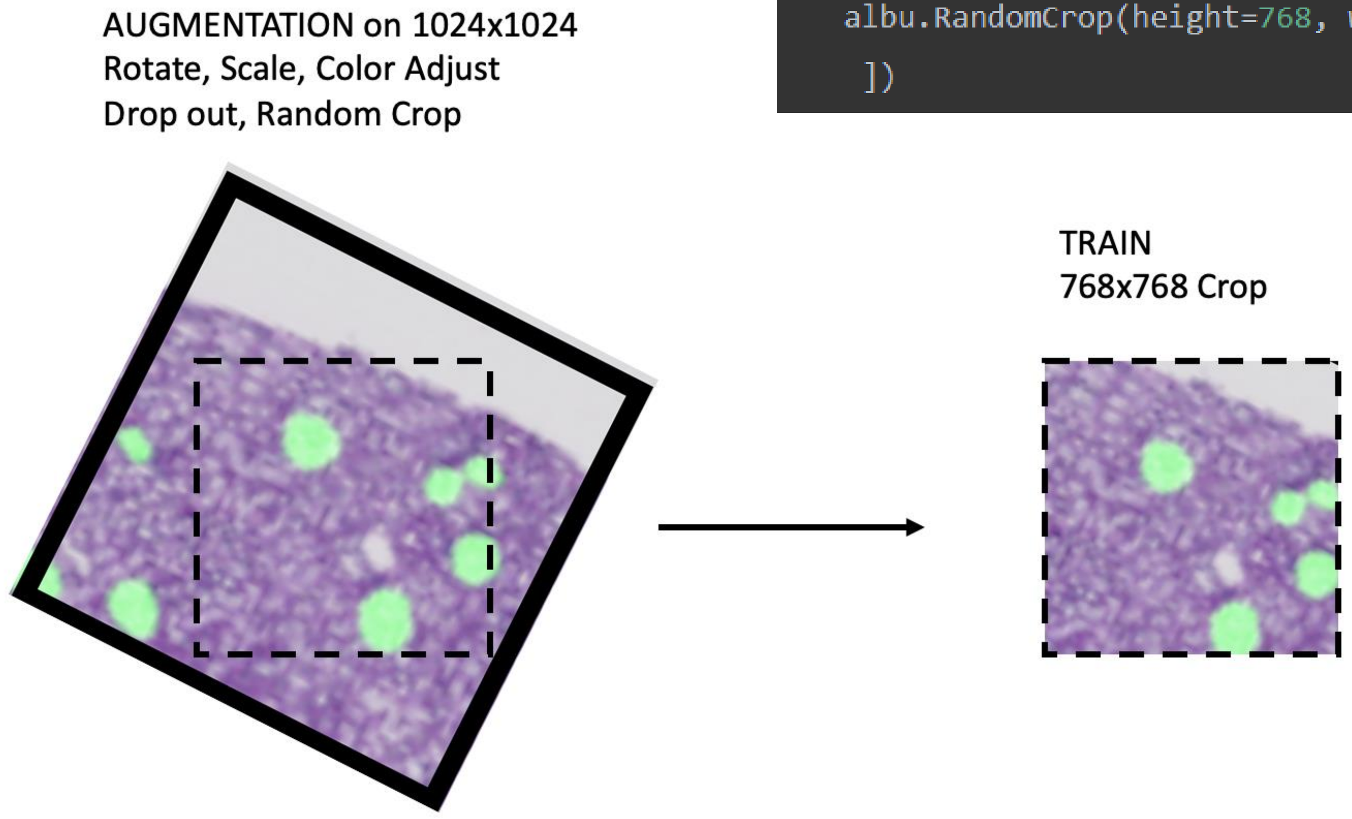
(1) 23rd Solution Summary

[Chris Solution]

- Augmentation



```
composition = albu.Compose([
    albu.HorizontalFlip(p=0.5),
    albu.VerticalFlip(p=0.5),
    albu.ShiftScaleRotate(rotate_limit=25, scale_limit=0.15, shift_limit=0, p=0.75),
    albu.CoarseDropout(max_holes=16, max_height=64 ,max_width=64 ,p=0.5),
    albu.ColorJitter(brightness=0.25, contrast=0.25, saturation=0.25, hue=0.25, p=0.75),
    albu.GridDistortion(num_steps=5, distort_limit=0.3, interpolation=1, p=0.5),
    albu.RandomCrop(height=768, width=768, p=1.0)
     ])
```



2. Solution

(1) 23rd Solution Summary

```
model.fit(train_gen(batch_size=24, image_size=(768,768)), epochs=50,
      callbacks=[lr], use_multiprocessing=True, workers=4)
model.save_weights('model_weights.h5')
```

[Chris Solution]

- FPN with Backbone EfficientNetB2 and BCE Jaccard Loss

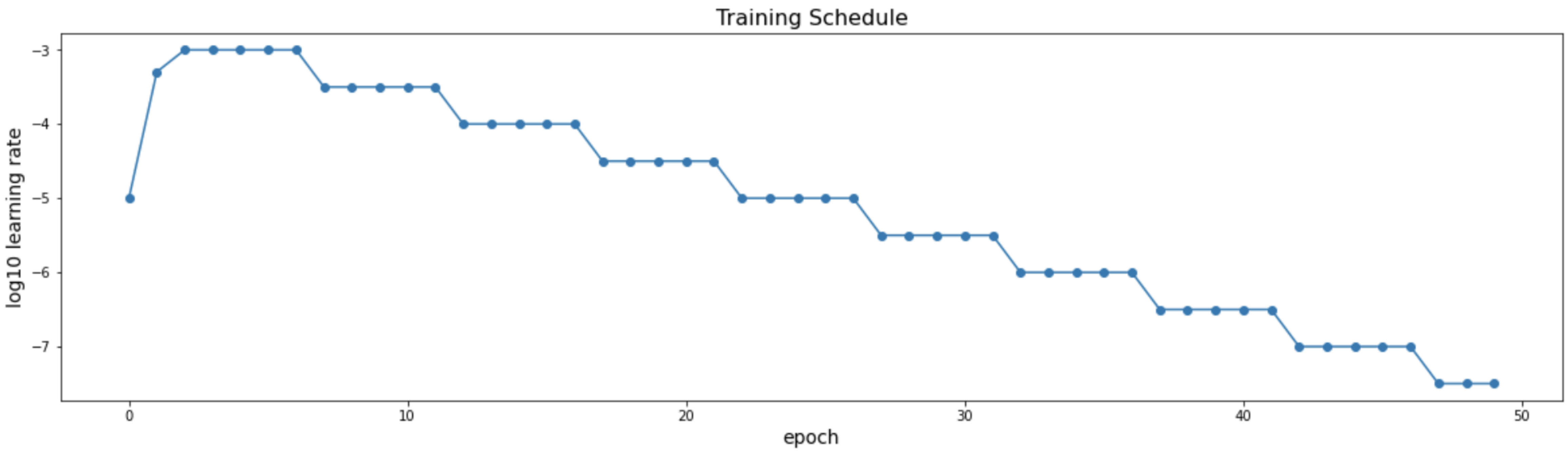
```
import segmentation_models as sm
# 768 x 768
def build_model():
    inp = tf.keras.Input(shape=(None,None,3))
    base = sm.FPN('efficientnetb2', encoder_weights='imagenet',
            classes=1, activation='sigmoid')
     x = base(inp)

    opt = tf.keras.optimizers.Adam()
    model = tf.keras.Model(inputs=inp, outputs=x)
    model.compile(
        optimizer=opt,
        loss=sm.losses.bce_jaccard_loss,
        metrics=[sm.metrics.f1_score]
    )
    return model
```

2. Solution

(1) 23rd Solution Summary

[Chris Solution]

- Training Schedule



2. Solution

(1) 23rd Solution Summary

[Chris Solution]
- Ensemble Strategy
- No Fold 100% Full Train Data

[Model]
- 1280x1280 reduce to1024x1024 with batch size 16

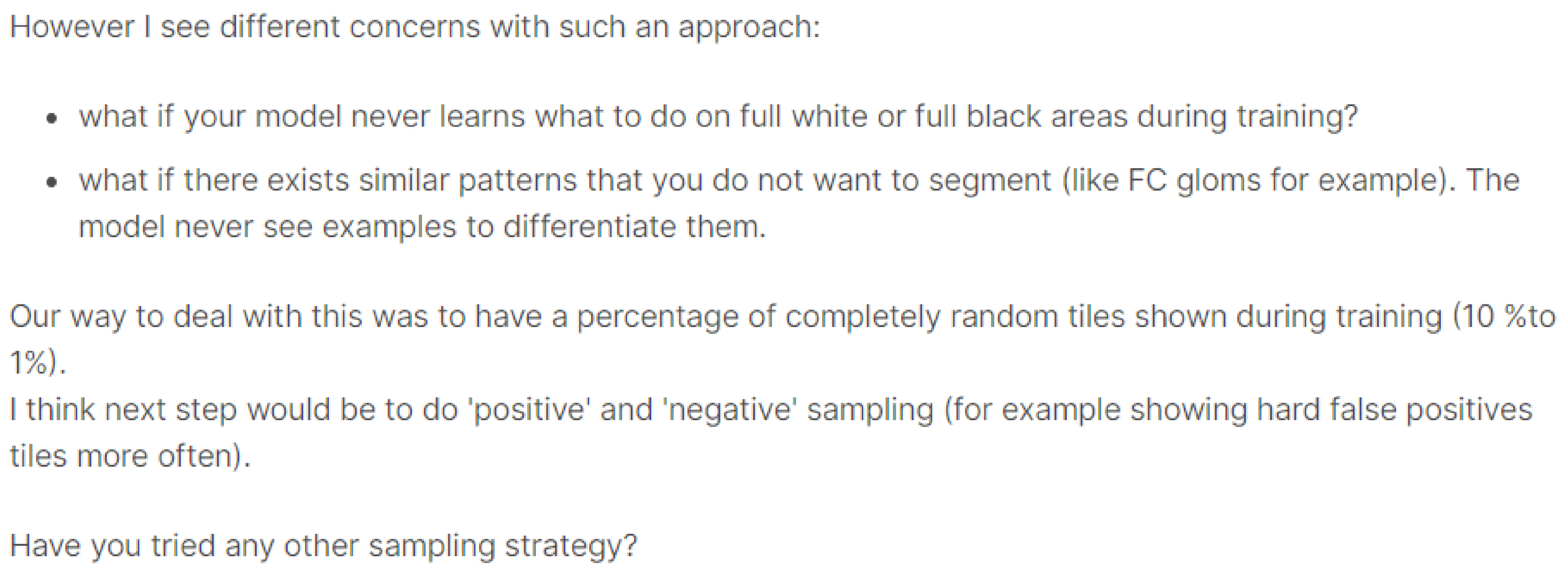
EfficientNetB0, EfficientNetB2
- 1 Model : 2x reduce images
- 1 Model : 3x reduce images
- 1 Model : 4x reduce images

2. Solution

(1) 23rd Solution Summary

[Chris Solution]

- Q&A에 대한 부분



However I see different concerns with such an approach:

- what if your model never learns what to do on full white or full black areas during training?
- what if there exists similar patterns that you do not want to segment (like FC gloms for example). The model never see examples to differentiate them.

Our way to deal with this was to have a percentage of completely random tiles shown during training (10 %to 1%).
I think next step would be to do 'positive' and 'negative' sampling (for example showing hard false positives tiles more often).

Have you tried any other sampling strategy?

2. Solution

(1) 23rd Solution Summary

[Chris Solution]

- I downloaded every public notebook submission.csv and mixed those in using genetic algorithms to maximize public LB

- I also randomly added and removed labeled gloms and found many annotator pattens and errors that were not discussed in the forums

- Instead, i ilteratively trained my model on hand label and pseudo labels. Each time it predicted better masks for d488c759a. I also used genetic algorithms to ensemble my model with public submission.csv files and made random changes to the mask. Eventually, i got my public LB up to 0.944

2. Solution

(1) 35th Solution Summary

[Segmentation & Yolo Models]

- Inference : https://www.kaggle.com/kevin1742064161/seg-yolo-final-inference?scriptVersionId=62406590

- Model : unet+b2, unet+b4, unetplusplus+b2, manet+b4
- window : 1024
- overlap : 100
- size : 512

- YOLOv5 is used to detect cells. By comparing the area of the bonding box with the area of the segmentation, we can judge whether the segmentation result needs to be filled