HuBMAP

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

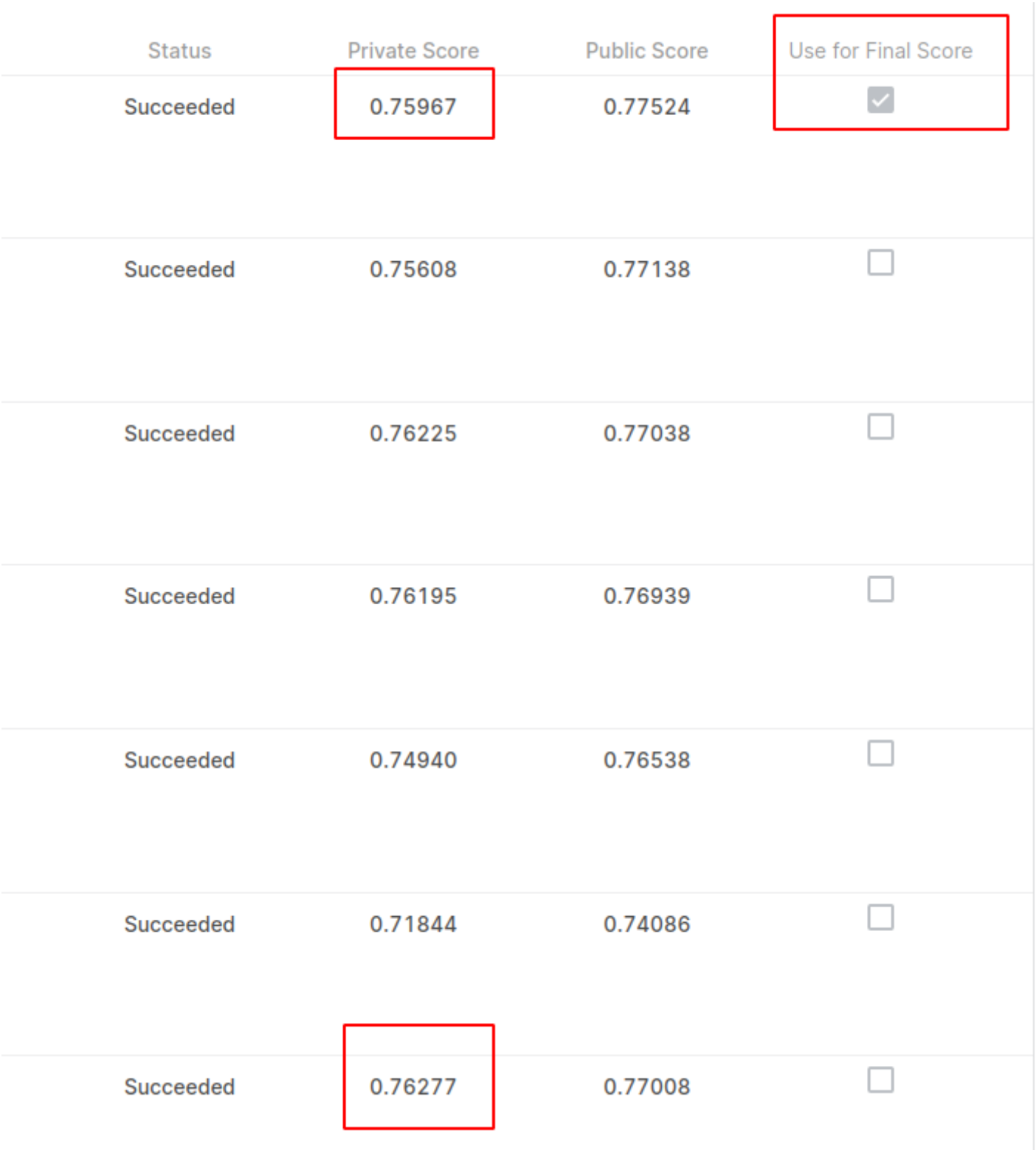
story==故事

很多人说写论文，就是写故事，那我想如果真的那么简单，那么我一定可以写的很好。
其实我现在一直很后悔一件事，那就是今年暑假我并没有去参加肠胃分割那个比赛，因为这两场比赛都是关于分割的，而且时间正好是一个结束一个开始，我透！如果当初不错过，这次也许就能拿牌了，惨兮兮。0.75967这是我成绩，铜牌线是0.76254。我可能想先讲个故事，在这个故事里是我参加这个比赛是各种想法，后面正文才是我对于一些知识整理。
毕竟是第一次参加这个比赛，一开始我真的满怀希望，况且我还报了个**之眼的比赛班，说实话，我当初想的就是，就算我可能不太行，但是这花钱报了班，它总会给代码，我总不至于成绩很差，也就可以把这段经历写在简历上。后来才知道这都是我一个人在yy。虽然成绩不是很好，但是我想我还是可以写在简历上的，哈哈。
那就从比赛刚开始说起，其实比赛很早就开始了，但是一开始没怎么有人呀，它这个比赛班也是在比赛还剩最后一个多月的时候开班的。其实这一点我也感觉到了，没必要说比赛一开始就去专注比赛，一个多月就够了，甚至有些厉害的人几个星期就很够了。然后我就上了第一节课，针对这个图像分割的比赛呢，（因为我是第一次参加比赛嘛，所以我就只是跟随比赛班的任务去做的，包括每节课布置的作业），一开始我们使用的就是经典的resnet模型，印象中当时我还在家，它的代码让我看的并不是那么简单，包括一些k-fold，还有模型为什么会有resnet50和resnet101等，我简单的看了代码，我不理解，但是我没有第一时间去debug，确实在家不是很方便，我相信绝大部分人拿到代码，第一件事就是去run，然后submit。况且我第一次参加，我还不会怎么submit。就这样我提交了自己的第一次。就是在去研究院的前一天，提交后lb400名，我就知道了路还很长。
对了，当时有一个作业实际上是替换模型，因为用到了resnet50和resnet101，作业是替换resnet50为efficientnet。然后我一脸懵逼？什么替换网络，我们都知道，替换模型很简单,把这个net替换了不叫好了。

```
model = net()
```

问题是什么，我要不要考虑输入输出的shape等等，就我们大家都知道一个人第一次做一件事，总会感觉无从下手。所以这件事其实我拖了很久。就是我一直在期待老师可以发一个代码，就是它替换好的代码是什么样的，这样我就可以对比一下，它是改了哪里。但是不幸的就是这件事没有，不可能，我才知道真的不会有人手把手教你，这个时候你才能体会到知识的价值。当然当时没做这件事还有一个原因：代码我都还不是很清晰，我好想没有看过这种k-fold，以及模型融合，还有tta等这些操作是干什么的，我不懂。所以我做的第一件事就是我要去debug。我相信，我debug之后了解了整个模型之后，我才能具备替换网络模型的能力。事实证明，就是这样的。

再后来我会更换网络模型之后，并且了解那一套代码之后，我就像一个小孩子一样，开始疯狂尝试，resnet50、resnet101、efficientnet各种融合，但是印象中分数最多也就是提升0.01而已。于是关于这个模型就止步于此了，但是我看讨论区里有人真的就使用efficientnet拿到了一个0.67左右的成绩，但是我却达不到，可能是因为使用的是efficientnet的小模型吧，就是b2，可能他们是使用的b7。(今天可能只能写story的部分了，关于后面的部分要等到明天了)所以他怎么做的，只能明天看了。之所以当时没看呢，是因为比赛班后来推荐我们使用swin transformer+upernet的一个模型，然后我明显感到就是我的电脑针对这个模型进行训练确实不太行，我渴望一张3090，后来有了一张3080才让我看到了一点点希望。是这样的，但是很明显batch_size到后面也是不行了。实际上呢，为什么比赛班会推荐这个模型，我基本已经get到了，实际上就是好像大家都是跟随青蛙哥（就是打比赛很厉害的一个人）的建议去做的。于是很简单，我就去跑了这个模型，然后成绩非但没有提高，而且还低的难以置信，我陷入了沉思，后来怎么说呢，错误在，我们在推理完之后，需要将mask进行一步resize，那么推理的数据的大小实际上并不是固定的，而我的代码中认为它是固定的（总结一下，这个错很zz）。再后来呢，我把输出的size改了之后呢，成绩依然不是很好，于是乎我开始调试阈值，因为我好想确实不太清楚应该调什么。在调了阈值之后呢，成绩终于有了提高，但是我不知道哇，为什么没有老师说的那么厉害，感觉我的提交成绩不太行呀。后来呢，因为课上老师说的是替换一下decoder，替换为segformer，对，其实很简单，我换了之后，提交成绩也不是很好，就在这里我彻底陷入了迷茫。后来也是看到讨论区有人说，resize的大小会提高一两个点，我起初是尝试直接改推理的代码，很明显这不切实际。后来我改训练代码，重新训练，但是问题来了，我的显存大小真的太小了，只有10G不太行，显存很容易就炸了。在这之间呢，。。。我破防了！刚才我打开了我的提交，出问题了呀！我选择了一个公榜成绩最高的作为提交成绩，但是这不是我私榜最高的成绩！而我私榜最高的成绩可以达到铜牌的的成绩，我透！！



Status	Private Score	Public Score	Use for Final Score
Succeeded	0.75967	0.77524	☑
Succeeded	0.75608	0.77138	☐
Succeeded	0.76225	0.77038	☐
Succeeded	0.76195	0.76939	☐
Succeeded	0.74940	0.76538	☐
Succeeded	0.71844	0.74086	☐
Succeeded	0.76277	0.77008	☐

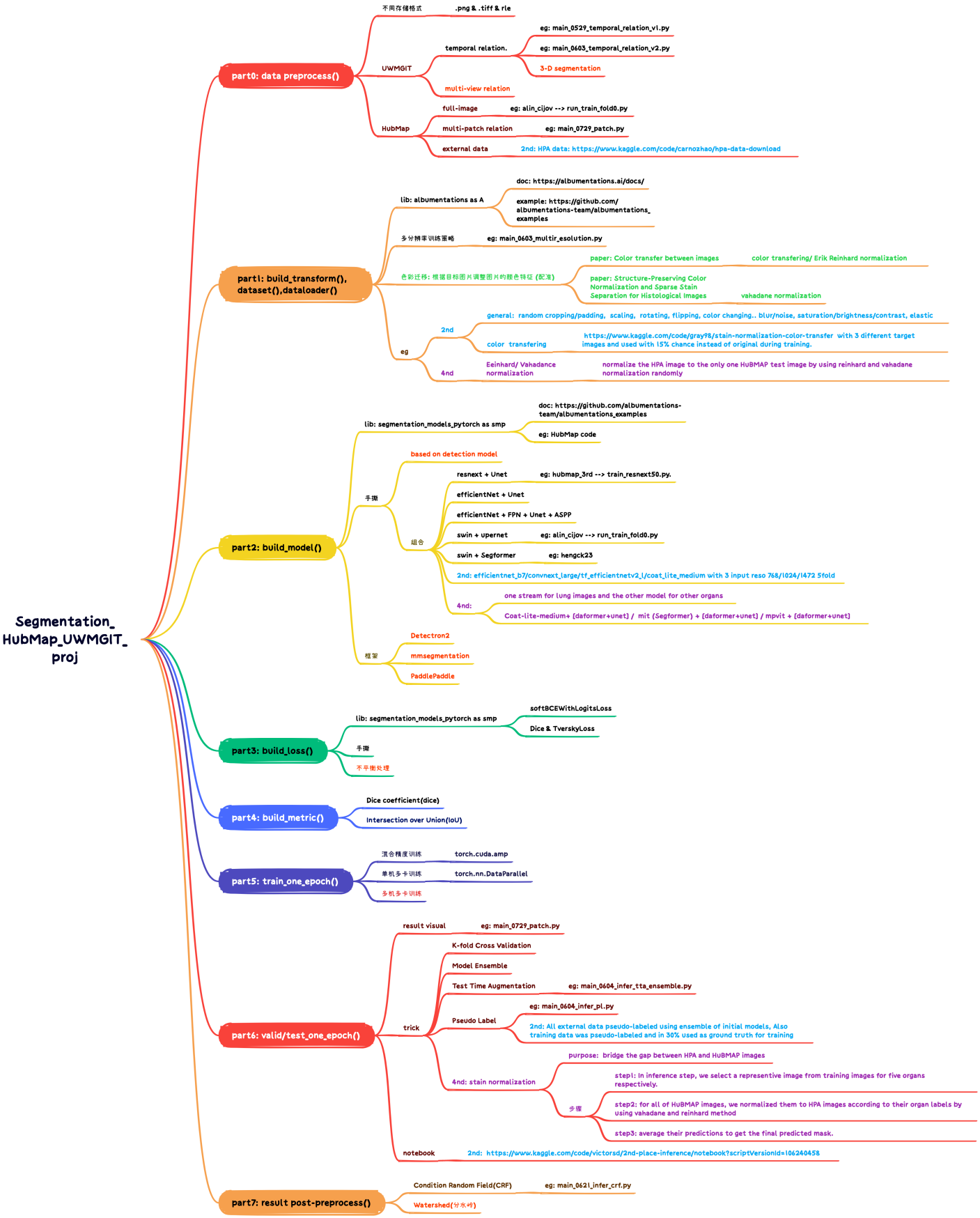
我只能说确实实力还是不行。

那我们继续吧，后来就是我不清楚为什么换了segformer的decoder，但是成绩却仍然不好，最后我换了一个模型也就是mitB2+segformer的模型，这个效果可就比之前的模型好太多了，也就是换了这个模型我开始上了0.7。然后就是各种调参，包括什么阈值呀，还有resize的大小呀，以及resize的mode，以及loss函数，以及推理时的tta，推理时进行的模型权重占比的设置。我印象中那个时候公榜成绩很不错了，后来就是我又尝试了新的模型coat，我也是这么开始一点点了解有关transformer的东西。但是

问题来了，coat我记得当时的问题是，模型收敛很慢，而且我电脑训练的也慢，最后感觉收敛很久，成绩都上不去。后来就是通过训练大图片，一点点提高成绩，但是我的显存不大，从768到896再到1024，我的显存就炸了。再后来呢就基本没怎么做了，原因呢首先是我感觉有点无从下手了，不太会做了。其次就是课题组有任务，我可能要去做了。但是其实我还是有一点想法，就是我知道可以把多个模型融合起来，但是确实有点无从下手，还有就是针对器官的种类进行训练，因为针对肺的推理成绩不太理想，所以可以针对肺使用一个模型进行训练，然后单独做推理，以及在讨论区可以看到就是针对不同器官的类别使用不同的阈值。其实这些我都懂，但是就是感觉无从下手，说的再简单点，就是我怎么用代码把我的想法实现是我们每个人都会面临的巨大问题。再后来就是等待比赛结束，真的太卷了，就是越到后面越卷，起初就是如果你一天没有提高，那么你的名词就会下降5名，越到后面呢。。。一言难尽。但是其实我们想到，最后成绩出来，我的私榜成绩会比公榜成绩高那么多名词，可能这就是k-fold以及tta的魅力把。好啦，我想故事就讲到这里的，明天就要去讨论区看看了，看看前面的code都是怎么做的。

正文

首先是一张总结的思维导图


Segmentation_HubMap_UWMGIT_proj
part0: data preprocess()
不同存储格式
.png & .tiff & rle
UWMGIT
temporal relation.
eg: main_0529_temporal_relation_v1.py
eg: main_0603_temporal_relation_v2.py
3-D segmentation
multi-view relation
HubMap
full-image
eg: alin_cijov --> run_train_fold0.py
multi-patch relation
eg: main_0729_patch.py
external data
2nd: HPA data: https://www.kaggle.com/code/carnozhao/hpa-data-download
part1: build_transform(), dataset(),dataloader()
lib: albumentations as A
doc: https://albumentations.ai/docs/
example: https://github.com/albumentations-team/albumentations_examples
多分辨率训练策略
eg: main_0603_multir_esolution.py
色彩迁移：根据目标图片调整图片的颜色特征 (配准)
paper: Color transfer between images
color transfering/ Erik Reinhard normalization
paper: Structure-Preserving Color Normalization and Sparse Stain Separation for Histological Images
vahadane normalization
eg
2nd
general: random cropping/padding, scaling, rotating, flipping, color changing.. blur/noise, saturation/brightness/contrast, elastic
color transfering
https://www.kaggle.com/code/gray98/stain-normalization-color-transfer with 3 different target images and used with 15% chance instead of original during training.
4nd
Eeinhard/ Vahadance normalization
normalize the HPA image to the only one HuBMAP test image by using reinhard and vahadane normalization randomly
part2: build_model()
lib: segmentation_models_pytorch as smp
doc: https://github.com/albumentations-team/albumentations_examples
eg: HubMap code
手撕
based on detection model
组合
resnext + Unet
eg: hubmap_3rd --> train_resnext50.py.
efficientNet + Unet
efficientNet + FPN + Unet + ASPP
swin + upernet
eg: alin_cijov --> run_train_fold0.py
swin + Segformer
eg: hengck23
2nd: efficientnet_b7/convnext_large/tf_efficientnetv2_l/coat_lite_medium with 3 input reso 768/1024/1472 5fold
4nd:
one stream for lung images and the other model for other organs
Coat-lite-medium+ [daformer+unet] / mit (Segformer) + [daformer+unet] / mpvit + [daformer+unet]
框架
Detectron2
mmsegmentation
PaddlePaddle
part3: build_loss()
lib: segmentation_models_pytorch as smp
softBCEWithLogitsLoss
Dice & TverskyLoss
手撕
不平衡处理
part4: build_metric()
Dice coefficient(dice)
Intersection over Union(IoU)
part5: train_one_epoch()
混合精度训练
torch.cuda.amp
单机多卡训练
torch.nn.DataParallel
多机多卡训练
part6: valid/test_one_epoch()
result visual
eg: main_0729_patch.py
trick
K-fold Cross Validation
Model Ensemble
Test Time Augmentation
eg: main_0604_infer_tta_ensemble.py
Pseudo Label
eg: main_0604_infer_pl.py
2nd: All external data pseudo-labeled using ensemble of initial models, Also training data was pseudo-labeled and in 30% used as ground truth for training
4nd: stain normalization
purpose: bridge the gap between HPA and HuBMAP images
步骤
step1: In inference step, we select a representive image from training images for five organs respectively.
step2: for all of HuBMAP images, we normalized them to HPA images according to their organ labels by using vahadane and reinhard method
step3: average their predictions to get the final predicted mask.
notebook
2nd: https://www.kaggle.com/code/victorsd/2nd-place-inference/notebook?scriptVersionId=106240458
part7: result post-preprocess()
Condition Random Field(CRF)
eg: main_0621_infer_crf.py
Watershed(分水岭)
Presented with xmind

数据预处理

对于这次比赛，数据预处理主要针对一下几件事情：
1、原始图片数据集是3000 *3000的 tiff的格式，需要提交的csv文件中需要将预测的mask图片转成rle编码。所以针对原始图片有切patch的。但是后面发现切patch的效果没有不切的好。所以到后面就不采用切patch的方法，而是使用resize的方法

2、比赛进行到后面，也有人提供了external data。那个时候我真的已经没怎么关注了这个比赛了。
在看了前几名的代码之后:没有人去分patch。原因可能是因为3000 * 3000并不是很大。
第二名使用了external data

数据增强，dataset，DataLoade于二分类和多类分割的都用了这个！

1、使用albumentations库，这个库的官网：https://albumentations.ai/docs/ 从代码中可以看出来，这里使用的数据增强按照train和val分为了两部分，train部分从resize和horizontalFlip选一个进行。只是一开始使用这个库的数据增强，后来就没有使用了，而是使用青蛙哥写的数据增强的代码。

```
import albumentation as A
def build_transforms(CFG):
    data_transforms = {
        "train": A.Compose([
            A.OneOf([
                A.Resize(*CFG.img_size, interpolation=cv2.INTER_NEAREST, p=1.0),
            ], p=1),
            A.HorizontalFlip(p=0.5),
            ], p=1.0),
        "valid_test": A.Compose([
            A.Resize(*CFG.img_size, interpolation=cv2.INTER_NEAREST),
            ], p=1.0)
        }
    return data_transforms
```

2、多分辨率训练的方案
因为类别说比较少，这个比赛只有五类，所以在这个比赛中work的效果不是很明显。
我找了一下代码，发现一个问题，就是实际上在我用的代码中没有使用到多分辨率训练的方案。

3、色彩迁移：根据目标图片调整图片的颜色配准
color transfering
vahadane normaization
传统图像处理？配准？具体了解要去看论文和前几名的代码

4、这里有人使用了cutmix或者cutout，看讨论区说是这种数据增强是可以涨点的，但是其实根据实际的理论分析，其实应该是不work的，甚至会降低整体的精确度。也就是这两个数据增强实际上主要是针对目标检测进行的。

第二名
general：random cropping/padding , scaling , rotating , flipping , color change,亮度增强，随机噪声增强

第四名

model

1、这里有一个库segmentation_models_pytorch
预训练---完形填空===transformer的预训练BERT如何无损的迁移到cv中。

2、在比赛中还有人使用了mmsegmentation和Detectron等
因为对框架的不够熟悉，所以没有使用。也是希望通过学习到网络的原始结构。

3、个人在比赛中对模型的选择调整

resnet+unet
将resnet换为提取特征能力更强的efficientnet-b0到b7（在模型中我只使用了b2，但是肯定是模型越大精确度越高的）
efficientnet+unet
但是特征能力仍然不足，所以在efficientnet和unet中间插入FPN结构，为了让每个pexi学到的感受野更大，在网络模型的最后加入了ASPP的模块
efficientnet+FPN+unet+ASPP
为进一步提高特征的提取，重新替换encoder和decoder
Swin+upernet
将upernet替换为基于transformer的分割模型segformer
swin+segformer
理论上这个模型的效果应该很好，但是在我实验中，实际上效果并不是很好，因此我将encoder部分换成了mit-b2
mit_b2+segformer
前面我们先不考虑名词靠前的人的代码是怎么做的，放到最后我们在看模型代码的。

loss

1、使用segmentation_models_pytorch库中的BCEWithLogitsLoss和Dice和TverskyLoss

2、更改loss函数为bce+dice，从代码中看，还有一个loss函数是aux_loss，这好像是一个辅助损失，实际上我们损失函数一般都是针对输出计算损失，然后反向传播的。但是这个损失应该是针对某一feature map进行反向传播。有关bce损失函数就说我们熟知的交叉熵损失函数，但是torch.nn.functional.binary_cross_entropy_with_logits和torch.nn.BCEWithLogitsLoss有啥区别呢？但是我们其实知道就是BCELoss和BCEWithLogitsLoss的区别就是多了一个sigmod的问题。针对Dice损失函数，

好像pytorch中没有封装dice损失这个函数吧，但是在segmentation_models_pytorch这个库中实际上是可以调用dice_loss的。

```
import torch.nn.functional as F
import segmentation_models_pytorch as smp
Dice_loss = smp.losses.DiceLoss(mode='binary')
#在net中调用损失函数，这里省略了一些细节
class Net(nn.Module):
        def load_pretrain( self,):
        pass
        def __init__( self,):
                super(Net, self).__init__()
        pass
        def forward(self, batch):
        output = {}
        if 'loss' in self.output_type:
            output['bce_loss'] = F.binary_cross_entropy_with_logits(logit,batch['mask'])
            output['dice_loss'] = Dice_loss(logit,batch['mask'])
            for i in range(4):
                output['aux%d_loss'%i] = criterion_aux_loss(self.aux[i](encoder[i]),batc
        if 'inference' in self.output_type:
            output['probability'] = torch.sigmoid(logit)
        return output

def criterion_aux_loss(logit, mask):
        mask = F.interpolate(mask,size=logit.shape[-2:], mode='nearest')
        loss = F.binary_cross_entropy_with_logits(logit,mask)
        return loss
#训练函数中调用模型，并且提取损失函数，将损失函数求和并做反向传播
output = net(batch)#这里的output只有loss
loss0  = output['bce_loss'].mean()
loss1  = output['aux2_loss'].mean()
loss2 = output['dice_loss'].mean()

optimizer.zero_grad()
scaler.scale(0.5*loss0+0.2*loss1+0.5*loss2+0.2*loss1).backward()#============这里改了loss
```

3、不平衡处理?

没有人使用，首先了解一下什么是不平衡loss，实际上就是针对数据集不平衡分布的loss函数，那么比赛中为什么没人使用就明白了，本次比赛的数据集不过才10个G，而且类别数也才五类，每个类分别均衡，不过就是存在不同类的识别难易程度不一致。所以也就不存在什么数据分布不平衡。针对不平衡loss主要使用的是Focal loss。Focal loss主要是为了解决one-stage目标检测中正负样本比例严重失衡的问题。该损失函数降低了大量简单负样本在训练中所占的权重，也可理解为一种困难样本挖掘。

metric

1、dice

有关这个评价指标的函数，大家使用的都是基本一样的，我看了下代码，dice函数实际上就是真实值的交集*2除以真实值的并集。接下来详细的看一下有关这两个指标的的区别和联系。dice和iou实际上都是衡量两个集合之间的相似性的度量。

$$dice(A,B)=\frac{2|A\cap B|}{|A|+|B|}$$

$$IOU(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

```
def compute_dice_score(probability, mask):
    N = len(probability)
    p = probability.reshape(N,-1)
    t = mask.reshape(N,-1)
    p = p>0.5
    t = t>0.5
    uion = p.sum(-1) + t.sum(-1)
    overlap = (p*t).sum(-1)
    dice = 2*overlap/(uion+0.0001)
    return dice
```

2、IOU

评价指标实际上就是在这两个中选择，所以我用了dice就没有再使用iou

train

1、混合精度训练

在pytorch的tensor中，默认训练的数据类型是float32，神经网络训练过程中，网络权重以及其他参数，默认数据类型都是float32，即单精度，为了节省内存，部分操作既有float32又有float16，因此叫做混合精度训练。

pytorch中是自动混合精度训练，使用 torch.cuda.amp.autocast 和 torch.cuda.amp.GradScaler 这两个模块。

torch.cuda.amp.autocast：在选择的区域中自动进行数据精度之间的转换，即提高了运算效率，又保证了网络的性能。

torch.cuda.amp.GradScaler：来解决数据溢出问题，即数据溢出问题：Overflow / Underflow

有关混合精度训练的使用

```
import torch.cuda.amp as amp
scaler = amp.GradScaler(enabled = is_amp)#自动混合精度，节省显 存并加快推理速度  scaler用于精度
net = Net().cuda()
net.train()
with amp.autocast(enabled = is_amp):
    # pdb.set_trace()
    output = net(batch)#这里的output只有loss
    loss0  = output['bce_loss'].mean()
    loss1  = output['aux2_loss'].mean()
    loss2 = output['dice_loss'].mean()

optimizer.zero_grad()
scaler.scale(0.5*loss0+0.2*loss1+0.5*loss2+0.2*loss1).backward()#============这里改了loss
scaler.unscale_(optimizer)
scaler.step(optimizer)
scaler.update()
```

2、单机多卡训练

```
torch.nn.DataParallel
```

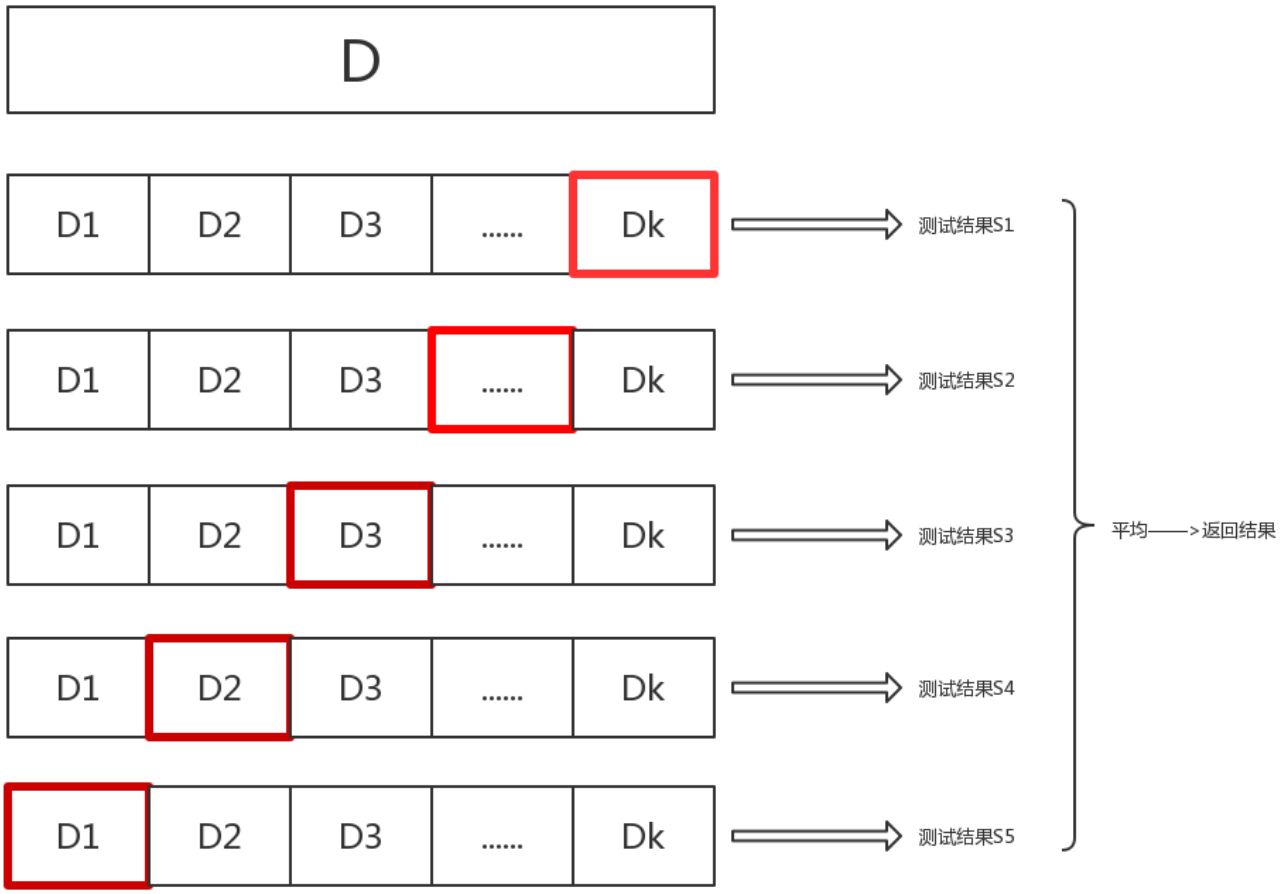
val&test

1、result visual于二分类和多类分割的
这里只是拿出了可视化的一个函数，主要问题实际上就是需要在哪个位置调用这个函数，以及注意这个函数的输入shape值，如果将其转换成plt想要的数据类型。然后可视化。还有一个问题就是，我有一个其他的模型在添加了可视化代码只有，发现明显运行速度会降低，也就是在运行时，会有内存的占用。所以我们需要注意把没用的删除掉。这也是我在这次比赛中学到的东
西del state_dict#？？？啥意思------删除的的意思。在本次比赛中呢，我的可视化基本没怎么使用，原因是一开始就不太懂，后面就明白了一个道理，就是这个可视化模块应该在一开始就添加到代码中，在val中调用这个函数，这样我们就可以了解到网络模型在学习的过程中有哪些问题。方便以后代码的改进。我是在最后一部分才添加了可视化代码，但是是在test中添加的，这就犯了一个大错，比赛中测试图片只有一张，并且没有真实值可以对比，所以可视化意义并不大。后来我意识到这一点之后，我尝试将训练的图片放在test的文件夹中，但也就是几张的可视化。所以在本次比赛中，可视化做的并不是很好，我会予以重视，尤其在后面一些项目中。

```python
def plot_visual(image, mask, pred, image_id, cmap):
    plt.figure(figsize=(16, 10))

    plt.subplot(1, 3, 1)
    plt.imshow(image.transpose(1, 2, 0))
    plt.grid(visible=False)
    plt.title("image", fontsize=10)
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(image.transpose(1, 2, 0))
    plt.imshow(mask.transpose(1, 2, 0), cmap=cmap, alpha=0.5)
    plt.title(f"mask", fontsize=10)
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(image.transpose(1, 2, 0))
    plt.imshow(pred.transpose(1, 2, 0), cmap=cmap, alpha=0.5)
    plt.title(f"pred", fontsize=10)
    plt.axis("off")

    plt.savefig(f"./result/{image_id}")
    # plt.show()
```

2、K-fold Cross Validation

K-fold交叉验证，是我在这个比赛中学到的第一个东西。一开始我们都是知道数据集的划分，也就是我们会将数据集划分为训练集、验证集、测试集。那么在k-fold中就是将数据分为n个部分，进行n次训练，每次训练都选择不同的部分作为验证集。那么在测试的时候应该怎么做呢，就是n次训练我们会产生n个权重，那么就需要把n个权重全部加载进来。进行n次测试，然后将每次的预测值求其平均值，就是最后的预测结果。有一个张图可以很好的说明这个事情。



这个是基操，老师说的。我的模型私榜成绩能提升20名我估计这个trick做了很大贡献，还有后面的tta。

```
from sklearn.model_selection import KFold
train_df, valid_df = make_fold(fold)
train_dataset = HubmapDataset(train_df, train_augment5b)
valid_dataset = HubmapDataset(valid_df, valid_augment5)

def make_fold(fold=0):
    # df = pd.read_csv('../input/hubmap-organ-segmentation/train.csv')
    df = pd.read_csv('/home/ktd/rpf_ws/HuBMAP/Code/input/hubmap-2022-256x256/train.csv')
    num_fold = 4
    skf = KFold(n_splits=num_fold, shuffle=True,random_state=42)#实例化，4fold，随机打乱，随
    df.loc[:,'fold']=-1#???给df增加一列，声明第几个fold，初始均为-1
    for f,(t_idx, v_idx) in enumerate(skf.split(X=df['id'], y=df['organ'])):#这里应该是根据
    #这里根据器官，可以理解为每个fold都有一定的不同器官的图片。
        df.iloc[v_idx,-1]=f#新添加的一列确定验证时为哪个fold，因为除去验证，其他的都是训练。所以只需
    train_df=df[df.fold!=fold].reset_index(drop=True)
    #df[df.fold!=fold]将fold的值变为T or F 、reset_index(drop=T)从新排序，之前的index去掉
    valid_df=df[df.fold==fold].reset_index(drop=True)
    return train_df,valid_df
```

3、Model Ensemble

有关模型的融合，说实话，我还不是很了解，但是在比赛过程中，我确实想过通过不同的模型对图片进行预测，最后求个平均值就可以。但是不太会做。这次比赛中，真正运用到的是，一个模型在训练时使用交叉验证产生的不同的权重文件，加载到模型，进行预测，最后求其平均值。但是这所有的权重都是一个网络模型的权重。我还是很想知道别人不同网络结构的模型是怎么融合到一起的。
下面是针对多个权重文件（相同的网络模型）的模型的融合和预测的代码：

```
for path in MODELS:#MODELS是一个列表，这里存放了各个权重存放的地址
    state_dict = torch.load(path,map_location=torch.device(device))
    if Ensemble_Weight:#这个是在加载不同pth文件时，我们可以通过设置他们的文件名，来确定不同pth所占比
        score = extract_model_score(path)
    else:
        score = 1
    all_score += score
    # model = Net(config=config()).to(device)
    model = Net().to(device)
    model.load_state_dict(state_dict["state_dict"])
    model.float()
    model.eval()
    model.to(device)
    models.append((model, score))
```

4、Test Time Augmentation

tta根据名字，首先了解到这是一个在测试时使用的数据增强。方法：测试时将原始数据做不同形式的增强，然后取结果的平均值作为最终结果。
作用：可以进一步提升最终结果的精度。

到这里我们很容易理解，像图像分类，图像分割，但是对于目标检测这个任务，还是那个问题，当图片进行旋转翻转，甚至裁剪等操作时，bbox应该如何变换，一定是有方法可以完成这个的，这也是我应该去学习的。当然在这次比赛中的tta的使用，我没有对其进行非常复杂的改变，而是简单的更改数据增强的方式。

```
if self.tta:
    # x,y,xy flips as TTA
    flips = [[-1], [-2], [-2, -1]]#flip按照维度进行反转
    for f in flips:
        xf = torch.flip(x, f)#torch.flip(变量，dim)
        for model, score in self.models:#因为多个权重，所以每个权重都要加载进来预测。这里的score
            p = model(xf)
            p = torch.flip(p, f)#预测完之后，再做一次翻转，得到原图的的预测结果
            py += p * score / self.all_score
    # pdb.set_trace()
    py /= (1 + len(flips))#一张图片实际预测了四组模型==这里的的1是因为上面还有一层对于原始图片的预测
```

但是我查资料发现，实际上tta也是可以进行调用的。

```
from pytorch_toolbelt.inference import tta
```

5、Pseudo label
一种简单高效的深度神经网络半监督学习方法。思想：将带标签和无标签的数据同时训练的监督方式。对于未标记的数据，伪标记，只是选取具有最大预测概率的类，当做他们是真实的标签一样使用。这实际上相当于熵正则化。倾向于类与类之间的低密度间隔。

结果后处理

本次比赛，实际上并没有使用什么后处理的方法。但是在课程中老师也提到了几个后处理方法。但是都没有使用，并且我自己本身不太了解，后续会继续学习补充。
1、CRF--condition random field
2、分水岭算法
步骤：将图像的所有像素按像素值从小到大排序，这里可以利用直方图将像素信息塞入数组。开始按灰度级从小到大顺序遍历所有像素，先将该灰度级的全部像素标记为待计算点。若点的邻域内有已存在的水池，则放入一个队列（先将水池边缘像素入队）。开始遍历队列，直到队列为空。开始计算下一个灰度级。参考：http://t.csdn.cn/embNg

针对前几名的分析

关于第二名，实际上第二名的代码看下来让我感触最深的就是模型的使用，难以想象他用了那么多模型，就是一开始在群里有人调侃使用了60个模型，我还觉着说着玩，现在看来确实是这样。还有就是感触很深的事情是，timm的这个库，确实对于一些比赛来说真的很重要。而且在csdn上去搜一些讲解，

但是不多甚至很少，这个时候我想大家都会知道官方文档的重要性了吧。看了他的代码只有我也是对于模型融合有了更深一层的认识了。首先让我们来看代码

```
#这是加载的一些参数，很明显可以看到，三个size的大小也就是他训练会分成了三个size，其实768，1024我们都可
params = [
    {'size': (768, 768), 'models': [
                                    ('tf_efficientnet_b7_ns', 'tf_efficientnet_b7_ns_768
                                    ('convnext_large_384_in22ft1k', 'convnext_large_384_
                                    ('tf_efficientnetv2_l_in21ft1k', 'tf_efficientnetv2_
                                    ('coat_lite_medium', 'coat_lite_medium_768_e40_{}_be
                                   ],
                        'pred_dir': 'test_pred_768', 'weight': 0.2},
    {'size': (1024, 1024), 'models': [
                                      ('convnext_large_384_in22ft1k', 'convnext_large_38
                                      ('tf_efficientnet_b7_ns', 'tf_efficientnet_b7_ns_1
                                      ('tf_efficientnetv2_l_in21ft1k', 'tf_efficientnetv
                                    ('coat_lite_medium', 'coat_lite_medium_1024_e41_{}_b
                                   ],
                        'pred_dir': 'test_pred_1024', 'weight': 0.3},
    {'size': (1472, 1472), 'models': [
                                    ('tf_efficientnet_b7_ns', 'tf_efficientnet_b7_ns_147
                                    ('tf_efficientnetv2_l_in21ft1k', 'tf_efficientnetv2_
                                    ('coat_lite_medium', 'coat_lite_medium_1472_e42_{}_b
                                   ],
                        'pred_dir': 'test_pred_1472', 'weight': 0.5},
]

class Timm_Unet(nn.Module):
    def __init__(self, name='resnet34', pretrained=True, inp_size=3, otp_size=1, decoder
        super(Timm_Unet, self).__init__()

        if name.startswith('coat'):
            #因为其他模型都是基于cnn的，但是coat实际上是基于transformer的，所以它的encoder要另外加
            encoder = coat_lite_medium()

            if pretrained:
                checkpoint = './weights/coat_lite_medium_384x384_f9129688.pth'
                checkpoint = torch.load(checkpoint, map_location=lambda storage, loc: st
                state_dict = checkpoint['model']
                encoder.load_state_dict(state_dict,strict=False)

            encoder_filters = encoder.embed_dims
        else:
            encoder = timm.create_model(name, features_only=True, pretrained=pretrained,
            #这里timm的库真的值得去花时间学习。
            encoder_filters = [f['num_chs'] for f in encoder.feature_info]
            #这一步的作用应该是获取encoder中每一层的输出通道数。方便后面做decoder的unet进行拼接前面

        decoder_filters = decoder_filters#这是decoder的每一层的通道数

        self.conv6 = ConvSilu(encoder_filters[-1], decoder_filters[-1])
        self.conv6_2 = ConvSilu(decoder_filters[-1] + encoder_filters[-2], decoder_filte
        self.conv7 = ConvSilu(decoder_filters[-1], decoder_filters[-2])
```

```
        self.conv7_2 = ConvSilu(decoder_filters[-2] + encoder_filters[-3], decoder_filte
        self.conv8 = ConvSilu(decoder_filters[-2], decoder_filters[-3])
        self.conv8_2 = ConvSilu(decoder_filters[-3] + encoder_filters[-4], decoder_filte
        self.conv9 = ConvSilu(decoder_filters[-3], decoder_filters[-4])

        if len(encoder_filters) == 4:
            self.conv9_2 = None
        else:
            self.conv9_2 = ConvSilu(decoder_filters[-4] + encoder_filters[-5], decoder_f

        self.conv10 = ConvSilu(decoder_filters[-4], decoder_filters[-5])

        self.res = nn.Conv2d(decoder_filters[-5], otp_size, 1, stride=1, padding=0)

        self.cls =  nn.Linear(encoder_filters[-1] * 2, 5)
        self.pix_sz =  nn.Linear(encoder_filters[-1] * 2, 1)

        self._initialize_weights()

        self.encoder = encoder

    def forward(self, x):
        batch_size, C, H, W = x.shape

        if self.conv9_2 is None:
            enc2, enc3, enc4, enc5 = self.encoder(x)
        else:
            enc1, enc2, enc3, enc4, enc5 = self.encoder(x)

        dec6 = self.conv6(F.interpolate(enc5, scale_factor=2))
        dec6 = self.conv6_2(torch.cat([dec6, enc4
                ], 1))

        dec7 = self.conv7(F.interpolate(dec6, scale_factor=2))
        dec7 = self.conv7_2(torch.cat([dec7, enc3
                ], 1))

        dec8 = self.conv8(F.interpolate(dec7, scale_factor=2))
        dec8 = self.conv8_2(torch.cat([dec8, enc2
                ], 1))

        dec9 = self.conv9(F.interpolate(dec8, scale_factor=2))

        if self.conv9_2 is not None:
            dec9 = self.conv9_2(torch.cat([dec9,
                    enc1
                    ], 1))

        dec10 = self.conv10(dec9) # F.interpolate(dec9, scale_factor=2))

        x1 = torch.cat([F.adaptive_avg_pool2d(enc5, output_size=1).view(batch_size, -1),
```

```
                        F.adaptive_max_pool2d(enc5, output_size=1).view(batch_size, -1)]

        # x1 = F.dropout(x1, p=0.3, training=self.training)
        organ_cls = self.cls(x1)
        pixel_size = self.pix_sz(x1)

        return self.res(dec10), organ_cls, pixel_size
models = []
for model_name, checkpoint_name, checkpoint_dir, model_weight in param['models']:
    for fold in range(5):
        model = Timm_Unet(name=model_name, pretrained=None)
        #到这里模型的整体结构就已经进来了，后面就是顺序加载模型的权重，然后将起放入列表models中
        snap_to_load = checkpoint_name.format(fold)
        print("=> loading checkpoint '{}'".format(snap_to_load))
        checkpoint = torch.load(path.join(checkpoint_dir, snap_to_load), map_location='c
        loaded_dict = checkpoint['state_dict']
        sd = model.state_dict()
        for k in model.state_dict():
            if k in loaded_dict:
                sd[k] = loaded_dict[k]
        loaded_dict = sd
        model.load_state_dict(loaded_dict)
        print("loaded checkpoint '{}' (epoch {}, best_score {})".format(snap_to_load,
            checkpoint['epoch'], checkpoint['best_score']))
        model = model.eval().cuda()

        models.append((model, model_weight))
#后面的预测环节就不写了，实际上就是把列表中的模型一个个取出来，然后将进行了tta的数据放入到模型中，最后得到
```

这就是第二名的代码，其他方面基本没有什么区别。对还有一个点实际上我不太明白，就是看了他的代码，他在数据进行预处理方面有一个函数，我不太懂。他对图像进行resize之后又进行了一个这个操作。我尝试性的搜了一下，真的有人说，也就是这个作者并没有将图片像素值取[0,1]而是选择了[-1,1]？目前我是这样理解的，但是不知道它的优势是啥。

```
#这是源码中每个模式不同的处理方式，可以看到'caffe'使用的为居中化，而 torch 与'tf'模式均为标准化，不同的
参考：http://t.csdn.cn/t1ALd


def preprocess_inputs(x):
    x = np.asarray(x, dtype='float32')
    x /= 127
    x -= 1
    return x
```

当然，第二名还有其他一些优化，比如他提供了一个额外的数据集、伪标签(代码中没看到，但是他自己说使用了)等。
后续有时间我还会多去看其他人的代码。

简历书写提示

简历写一些解决方案
数据集是什么，目的是什么
transformer模型
数据增强用了什么
从分辨率，以病理颜色增强，自适应的？？大小的数据增强。
采用？？？model
采用？？？loss===>如果你写了focal loss,面试会问为什么选择focal loss，举个例子哪个类占得比例低，你的系数是怎么调的?
tta
伪标签
后处理
数据集--->transformer--->
数据集是什么+增强+模型+loss+推理+指标
+个人博客地址
github repo

技能特长
每个项目都要有描述自己做了什么
数据集有多少张，最终的metric多少

目标检测很有用?
写一些你懂的，有把握的，不要写有歧义的
出生年月？籍贯？政治面貌？计算机二级？===不需要