1. Resize the images and labels in the dataset to 512x512
2. Generate ground truth key-patch map

What if I use kpm\_gen function to get the point\_heatmap every time I load the data and run my main code?

My current dataset is SIIM-ACR Pneumothorax Segmentation Competition 2019 dataset. My task is segmentation of pneumothorax from chest x-rays. I want you to modify the process\_point.py code to generate a new set of generated ground truth key-patch map for my dataset not the isic2016 or isic2018 datasets. First resize the images and masks in my current dataset to 512x512 from 1024x1024 like it's done in the process\_resize.py file, you can skip the division by 255 part. Do use np.clip(x, 0, 255). Then, in the kpm\_gen function, change the R and N parameters according to what is suitable for my dataset and task.

"C:/Users/nisha/Desktop/Research Project May-June/train\_png"

"C:/Users/nisha/Desktop/Research Project May-June/mask"

"C:/Users/nisha/Desktop/Research Project May-June/gt\_keypatch"

from torchvision.models import resnet50, resnet34, resnet18, ResNet50\_Weights, ResNet18\_Weights, ResNet34\_Weights

def ResNet18\_OS16(multi\_scale=False):

    return resnet18(weights=ResNet18\_Weights.IMAGENET1K\_V1)

def ResNet50\_OS16(multi\_scale=False):

    return resnet50(weights=ResNet50\_Weights.IMAGENET1K\_V2)

import torch

import torch.nn as nn

import torch.nn.functional as F

import os

import sys

# class DeepLabV3(nn.Module): | "DeepLabV3" is called as "base" in the BAT class

class base(nn.Module):

    def \_\_init\_\_(self, num\_classes, num\_layers):

        super(base, self).\_\_init\_\_()

        self.num\_classes = num\_classes

        layers = num\_layers

ASPP\_Bottleneck(num\_classes=self.num\_classes) instead

        if layers == 18:

            self.resnet = ResNet18\_OS16()

            self.aspp = ASPP(num\_classes=self.num\_classes)

        elif layers == 50:

            self.resnet = ResNet50\_OS16()

            self.aspp = ASPP\_Bottleneck(num\_classes=self.num\_classes)

    def forward(self, x):

        # (x has shape (batch\_size, 3, h, w))

        h = x.size()[2]

        w = x.size()[3]

        feature\_map = self.resnet(x)

        # (shape: (batch\_size, 512, h/16, w/16)) (assuming self.resnet is ResNet18\_OS16 or ResNet34\_OS16.

        # If self.resnet is ResNet18\_OS8 or ResNet34\_OS8, it will be (batch\_size, 512, h/8, w/8).

        # If self.resnet is ResNet50-152, it will be (batch\_size, 4\*512, h/16, w/16))

        output = self.aspp(

            feature\_map)  # (shape: (batch\_size, num\_classes, h/16, w/16))

        output = F.upsample(

            output, size=(h, w),

            mode="bilinear")  # (shape: (batch\_size, num\_classes, h, w))

        return output

model = BAT(1, 50, 1, 6).cuda()

def train\_one\_epoch(train\_loader, model, optimizer, loss\_fn, accumulation\_steps=int(EFFECTIVE\_BATCH\_SIZE/BATCH\_SIZE), device='cuda'):

    losses = AverageMeter()

    # Lists to store batch-to-batch progress details within the epoch while training

    batch\_count\_train = []

    batch\_train\_loss = []

    model = model.to(device)

    model.train()

    if accumulation\_steps > 1:

      optimizer.zero\_grad()

    tk0 = tqdm(train\_loader, total=len(train\_loader))

    for b\_idx, data in enumerate(tk0):

      # print(data['image'].shape) -> torch.Size([8, 3, 512, 512])

      # print(data['mask'].shape) -> torch.Size([8, 1, 512, 512])

      if (b\_idx + 1) % accumulation\_steps == 0:

        batch\_count\_train.append(b\_idx)

      # moves image tensor and mask tensor to gpu

      for key, value in data.items():

        data[key] = value.to("cuda")

      point = (data['point'] > 0).cuda().float()

      if parse\_config.net\_layer == 18:

          point\_c4 = nn.functional.max\_pool2d(point,

                                              kernel\_size=(16, 16),

                                              stride=(16, 16))

          point = nn.functional.max\_pool2d(point,

                                          kernel\_size=(8, 8),

                                          stride=(8, 8))

      else:

          point\_c5 = nn.functional.max\_pool2d(point,

                                              kernel\_size=(32, 32),

                                              stride=(32, 32))

          point\_c4 = nn.functional.max\_pool2d(point,

                                              kernel\_size=(16, 16),

                                              stride=(16, 16))

      if accumulation\_steps == 1 and b\_idx == 0:

        optimizer.zero\_grad()

      if parse\_config.point\_pred == 1:

            output, point\_maps\_pre = model(data['image'])

            output = torch.sigmoid(output)

            #print("point\_pre shape:{}, point shape:{}".format(point\_pre.shape,point.shape))

            assert (output.shape == data['mask'].float().shape)

            loss\_dc = dice\_loss(output, data['mask'].float())

            # print(point\_maps\_pre[-1].shape, point\_c4.shape)

            assert (point\_maps\_pre[-1].shape == point\_c4.shape)

            point\_loss = 0.

            for i in range(len(point\_maps\_pre)):

                point\_loss += criterion(point\_maps\_pre[i], point\_c4)

            point\_loss = point\_loss / len(point\_maps\_pre)

            loss = loss\_dc + point\_loss  # point\_loss weight: 3

            with torch.set\_grad\_enabled(True):

              loss.backward()

              # if (b\_idx + 1) % accumulation\_steps == 0:

              #   if GRADIENT\_CLIPPING:

              #     clip\_grad\_norm\_(model.parameters(), GRADIENT\_CLIPPING\_THRESHOLD)

              optimizer.step()

              optimizer.zero\_grad()

            iteration = iteration + 1

            if (b\_idx + 1) % 10 == 0:

                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

                    b\_idx \* len(data), len(train\_dataloader.dataset),

                    100. \* b\_idx / len(train\_dataloader), loss.item()))

      print("Iteration numbers: ", iteration)

      losses.update(loss.item(), train\_loader.batch\_size)

      if (b\_idx + 1) % accumulation\_steps == 0:

        batch\_train\_loss.append(loss.item())

      tk0.set\_postfix(loss=losses.avg, learning\_rate=optimizer.param\_groups[0]['lr'])

    return losses.avg, batch\_count\_train, batch\_train\_loss

Error:  
 **11** for epoch in range(1, till\_epoch+1):

**12** epoch\_count.append(epoch)

---> 13 loss, batch\_count\_train, batch\_train\_loss = train\_one\_epoch(train\_dataloader, model, optimizer, criterion)

**14** loss\_values.append(loss)

**15** store\_batch\_training\_details(epoch, batch\_count\_train, batch\_train\_loss)

<ipython-input-47-53e957744552> in train\_one\_epoch(train\_loader, model, optimizer, loss\_fn, accumulation\_steps, device)

**50** loss\_dc = dice\_loss(output, data['mask'].float())

**51** # print(point\_maps\_pre[-1].shape, point\_c4.shape)

---> 52 assert (point\_maps\_pre[-1].shape == point\_c4.shape)

**53**

**54** point\_loss = 0.

AssertionError: