unet-kl

November 9, 2022

0.1 prepare data

20 subjects in total, 18 will be used for training and validation, 2 will be used for independent testing raw images in nifti format, conver to png format, in order to use the existing augmentation solution.

```
[1]: import os
     import numpy as np
     import nibabel as nib
     import matplotlib.pyplot as plt
     import torch.nn as nn
     from PIL import Image
```

```
[2]: from fastai.data.all import *
     from fastai.vision.all import *
```

```
[3]: #nii folder has orig/label images in nifti format, orig and label would have␣
     ↪corresponding images in png formats
     train_path = Path('./data/train/')
     test_path = Path('./data/test/')
```

```
[4]: def nii2png(nii_path, png_path_orig, png_path_label):
         '''convert nii images to png files, and save original images to␣
     ↪png_path_orig, and labeled  images to png_path_label'''
         os.system('if [ ! -d {imgdir} ]; then mkdir -p {imgdir} ; fi;'.
     ↪format(imgdir = png_path_orig))
         os.system('if [ ! -d {imgdir} ]; then mkdir -p {imgdir} ; fi;'.
     ↪format(imgdir = png_path_label))

         nifti_images = sorted(get_files(nii_path,extensions='.gz')) #all nifti␣
     ↪images under folder: data/train/nii

         label_images = nifti_images[::2]
         raw_images = nifti_images[1::2]


         for (raw,label) in zip(raw_images, label_images):
             sub_name = raw.stem[:10] #substr, e.g., ircad_e01_
```

```
        raw_data = nib.load(raw,).get_fdata().astype(np.uint8) #intensity
    ↪within [0.255]
        label_data = nib.load(label).get_fdata().astype(np.uint8)

        assert raw_data.shape == label_data.shape

        print('working on subject : {}'.format(sub_name[:-1]))
        for idx in range(raw_data.shape[2]):
            orig_name = sub_name + '{:03}.png'.format(idx) #e.g., ircad_e01_000.
    ↪png
            label_name = sub_name + '{:03}L.png'.format(idx) #e.g.,
    ↪ircad_e01_000L.png
            Image.fromarray(raw_data[:,:,idx]).save(png_path_orig/orig_name)
            Image.fromarray(label_data[:,:,idx]).save(png_path_label/label_name)
```

```
[5]: # for training and validation;
    nii2png(nii_path = train_path/'nii',png_path_orig = train_path/'orig',
      ↪png_path_label=train_path/'label')
```

```
working on subject : ircad_e01
working on subject : ircad_e02
working on subject : ircad_e03
working on subject : ircad_e04
working on subject : ircad_e05
working on subject : ircad_e06
working on subject : ircad_e07
working on subject : ircad_e08
working on subject : ircad_e09
working on subject : ircad_e10
working on subject : ircad_e11
working on subject : ircad_e12
working on subject : ircad_e13
working on subject : ircad_e14
working on subject : ircad_e15
working on subject : ircad_e16
working on subject : ircad_e17
working on subject : ircad_e18
```

```
[6]: # for independent testing;
    nii2png(nii_path = test_path/'nii',png_path_orig = test_path/'orig',
      ↪png_path_label=test_path/'label')
```

```
working on subject : ircad_e19
working on subject : ircad_e20
```

0.2 verify images

```
[7]: def show_pair_overlay(seg, raw, alpha = 0.7):
         f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
         ax1.imshow(raw, cmap='gray')
         ax2.imshow(raw, cmap='jet')
         ax2.imshow(seg, cmap='gray', interpolation='none', alpha=alpha)
         return f, (ax1, ax2)
```
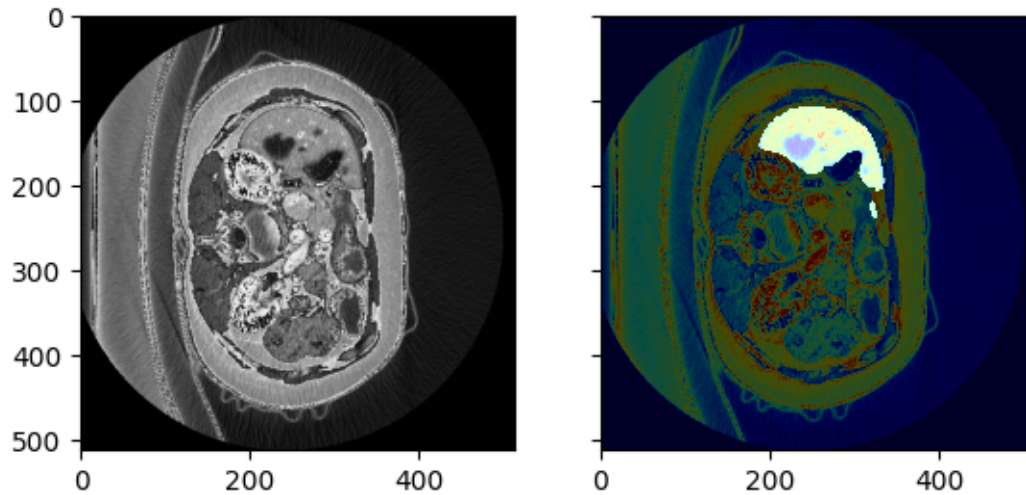
```
[8]: #verify training dataset
     orig = Image.open('data/train/orig/ircad_e01_104.png');
     label = Image.open('data/train/label/ircad_e01_104L.png')
     show_pair_overlay(label, orig, alpha = 0.7)
```

```
[8]: (<Figure size 640x480 with 2 Axes>, (<AxesSubplot: >, <AxesSubplot: >))
```
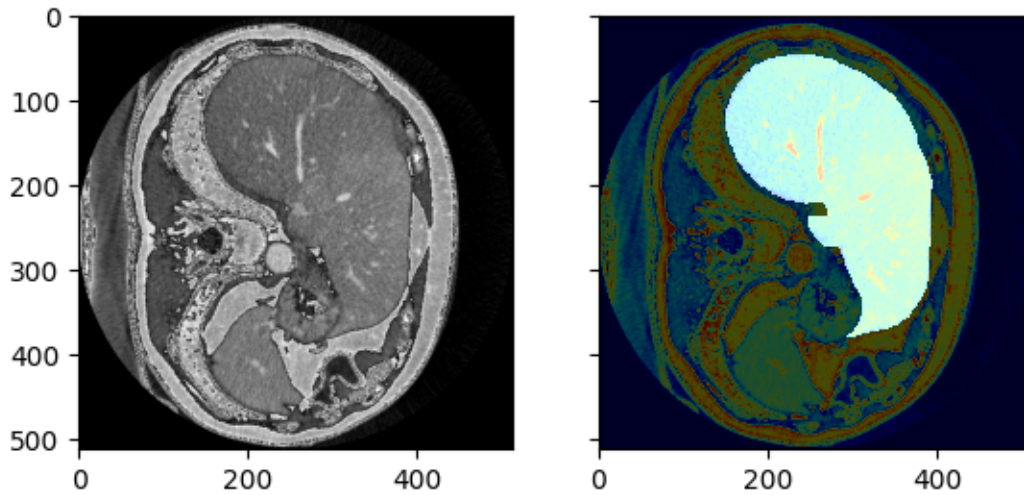


```
[9]: #verify testing dataset
     orig = Image.open('data/test/orig/ircad_e19_041.png');
     label = Image.open('data/test/label/ircad_e19_041L.png')
     show_pair_overlay(label, orig, alpha = 0.7)
```

```
[9]: (<Figure size 640x480 with 2 Axes>, (<AxesSubplot: >, <AxesSubplot: >))
```

0.3 DataLoaders

```
[141]: path = Path('./')
       dls = SegmentationDataLoaders.from_label_func(
           path, bs=4, fnames = get_image_files(path/'data/train/orig'),
           label_func = lambda o: path/'data/train/label'/f'{o.stem}L{o.suffix}',
           #liver images are put at data/label, with 'L' appended to the original name
           img_cls=PILImageBW,#only one channel, gray, not rgb;
           item_tfms=RandomResizedCrop(256, min_scale=0.3), # works on each pair of
           (orig,label)
           batch_tfms=aug_transforms(flip_vert=True) # works on each batch
       )
```

```
[146]: dls.show_batch()
```
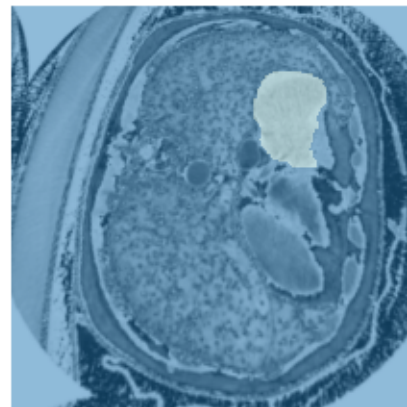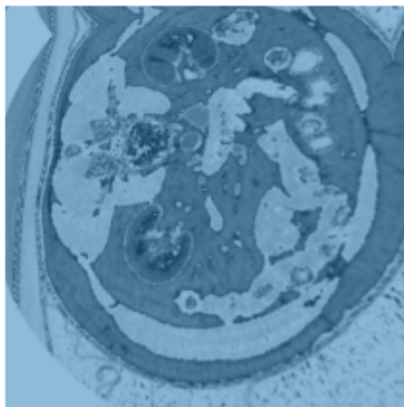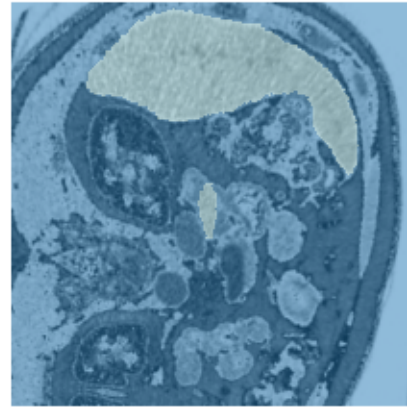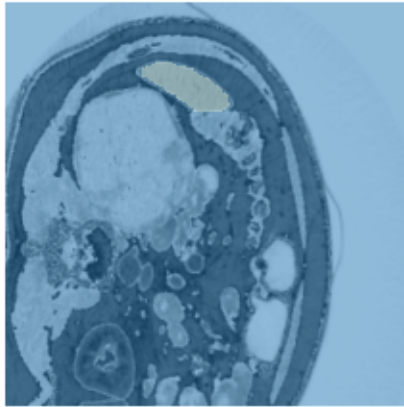
```
[35]: #check batch dims
      x,y = dls.train.one_batch()
      x.shape, y.shape
```

```
[35]: (torch.Size([4, 1, 256, 256]), torch.Size([4, 256, 256]))
```

0.4 unet with pretrained resnet

```
[153]: loss_func = CrossEntropyLossFlat(axis=1) #softmax over the channel dim
       learn = unet_learner(dls, resnet34, n_in = 1, n_out=2, opt_func = Adam,
        ↪loss_func = loss_func, metrics=Dice())
```

```
[16]: learn.model
```

```
[16]: DynamicUnet(
        (layers): ModuleList(
          (0): Sequential(
            (0): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
```

```
bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
      (4): Sequential(
        (0): BasicBlock(
          (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
        (1): BasicBlock(
          (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
        (2): BasicBlock(
          (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (5): Sequential(
        (0): BasicBlock(
          (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (6): Sequential(
    (0): BasicBlock(
```

```
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (3): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
      (4): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (5): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
```

```
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
  )
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (2): ReLU()
  (3): Sequential(
    (0): ConvLayer(
      (0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (1): ReLU()
    )
    (1): ConvLayer(
      (0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (1): ReLU()
    )
  )
  (4): UnetBlock(
    (shuf): PixelShuffle_ICNR(
      (0): ConvLayer(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU()
      )
      (1): PixelShuffle(upscale_factor=2)
    )
    (bn): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): ConvLayer(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
```

```
    (conv2): ConvLayer(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (relu): ReLU()
  )
  (5): UnetBlock(
    (shuf): PixelShuffle_ICNR(
      (0): ConvLayer(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU()
      )
      (1): PixelShuffle(upscale_factor=2)
    )
    (bn): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): ConvLayer(
      (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (conv2): ConvLayer(
      (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (relu): ReLU()
  )
  (6): UnetBlock(
    (shuf): PixelShuffle_ICNR(
      (0): ConvLayer(
        (0): Conv2d(384, 768, kernel_size=(1, 1), stride=(1, 1))
        (1): ReLU()
      )
      (1): PixelShuffle(upscale_factor=2)
    )
    (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv1): ConvLayer(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (conv2): ConvLayer(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (relu): ReLU()
  )
  (7): UnetBlock(
```

```
(shuf): PixelShuffle_ICNR(
  (0): ConvLayer(
    (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
  )
  (1): PixelShuffle(upscale_factor=2)
)
(bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(conv1): ConvLayer(
  (0): Conv2d(192, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
)
(conv2): ConvLayer(
  (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
)
(relu): ReLU()
)
(8): PixelShuffle_ICNR(
  (0): ConvLayer(
    (0): Conv2d(96, 384, kernel_size=(1, 1), stride=(1, 1))
    (1): ReLU()
  )
  (1): PixelShuffle(upscale_factor=2)
)
(9): ResizeToOrig()
(10): MergeLayer()
(11): ResBlock(
  (convpath): Sequential(
    (0): ConvLayer(
      (0): Conv2d(97, 97, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
    )
    (1): ConvLayer(
      (0): Conv2d(97, 97, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
  )
  (idpath): Sequential()
  (act): ReLU(inplace=True)
)
(12): ConvLayer(
  (0): Conv2d(97, 2, kernel_size=(1, 1), stride=(1, 1))
)
(13): fastai.layers.ToTensorBase(tensor_cls=<class
'fastai.torch_core.TensorBase'>)
)
```

```
                    )

[144]:  learn.fine_tune(10)

        <IPython.core.display.HTML object>

        <IPython.core.display.HTML object>

        <IPython.core.display.HTML object>

        <IPython.core.display.HTML object>

[145]:  learn.recorder.plot_loss() #seems to converge and a few more epochs would be␣
        ↪better.
```
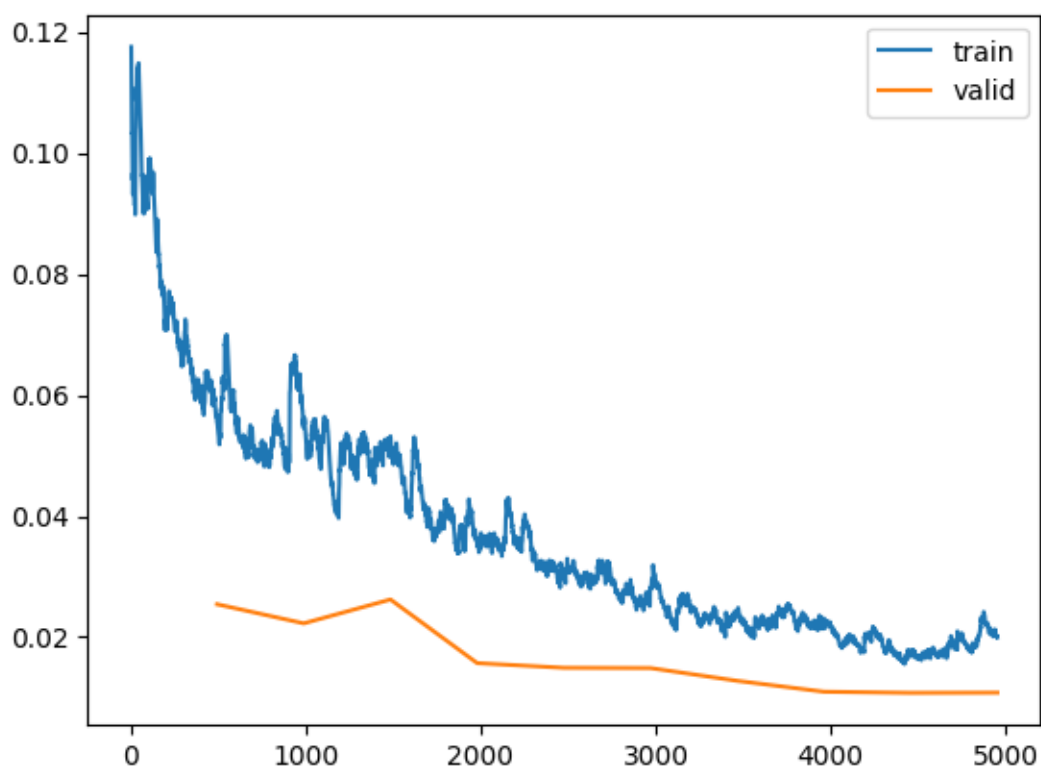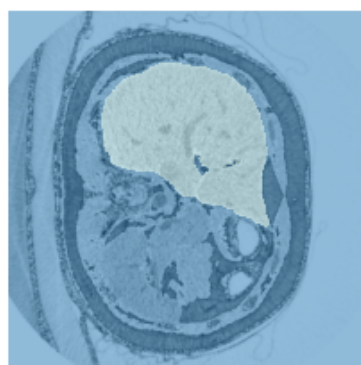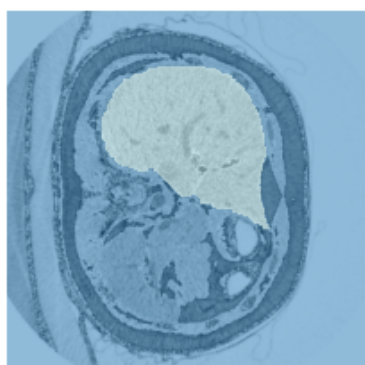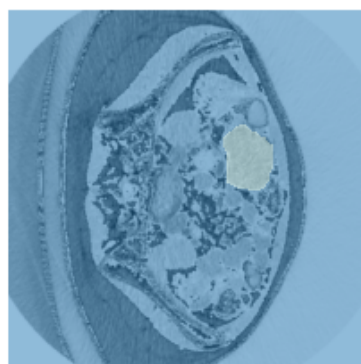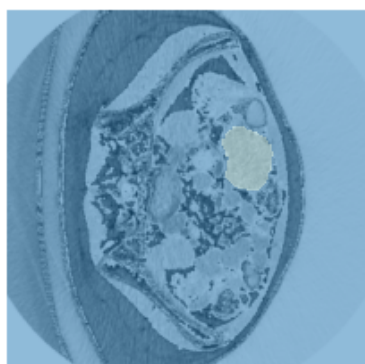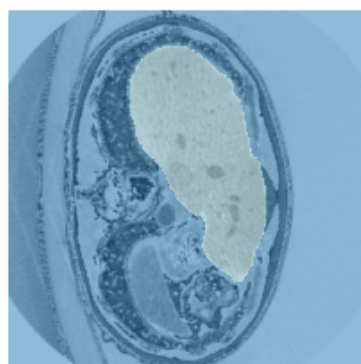


```
[23]:  learn.show_results()

       <IPython.core.display.HTML object>

       <IPython.core.display.HTML object>
```
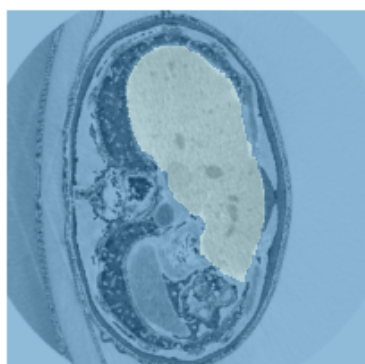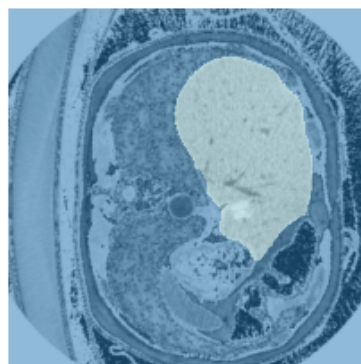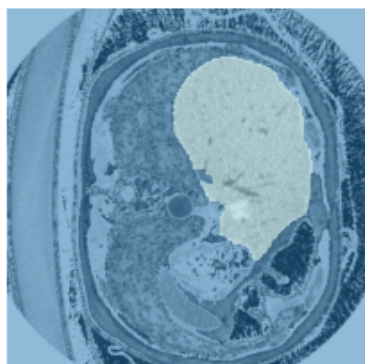
```
[5]: mdl_name = 'unet-resnet34-liverseg'
```

```
[148]: learn.save(mdl_name) # target: path/models/mdl_name
```

```
[148]: <fastai.learner.Learner at 0x7fdd57c98c70>
```

0.5 predict on independent testing dataset

```
[24]: def dice_coef(y_true, y_pred, eps=1e-8):
          y_true_f = torch.flatten(y_true)
          y_pred_f = torch.flatten(y_pred)
          intersection = torch.sum(y_true_f * y_pred_f)
          return ((2. * intersection + eps) / (torch.sum(y_true_f) + torch.
       ↪sum(y_pred_f) + eps)).item()
```

```
[6]: path = Path('./')
     dlsT = SegmentationDataLoaders.from_label_func(
         path, bs=4, fnames = get_image_files(path/'data/test/orig'),
         valid_pct = 0.99, #
         label_func = lambda o: path/'data/test/label'/f'{o.stem}L{o.suffix}',
         img_cls=PILImageBW,#only one channel, gray, not rgb;
         #item_tfms=Resize(256, method=ResizeMethod.Squish), # change to this resize␣
       ↪reduces the performance slightly
         item_tfms=RandomResizedCrop(256, min_scale=0.3), # works on each pair of␣
       ↪(orig,label)
     )
```

```
[7]: loss_func = CrossEntropyLossFlat(axis=1) #softmax over the channel dim
     test = unet_learner(dlsT, resnet34, n_in = 1, n_out=2, opt_func = Adam,␣
       ↪loss_func = loss_func, metrics=Dice()) #
     test.load(mdl_name)
```

```
/home/kli/miniconda3/envs/tf/lib/python3.9/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/kli/miniconda3/envs/tf/lib/python3.9/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=ResNet34_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet34_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
[7]: <fastai.learner.Learner at 0x7f7b701f2190>
```

```
[8]: test.freeze()
```

```
[9]: _, test_dice = test.validate()
     print('independent testing subjects, dice: {:6f}'.format(test_dice))
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

independent testing subjects, dice: 0.951924

```
[38]: test.show_results()
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Target/Prediction