```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount,
```

```
!python --version
```

```
Python 3.12.11
```

```
import os
os.getcwd()
```

```
'/content/drive/My Drive/XRA'
```

```
os.chdir('/content/drive/MyDrive/XRA')
os.getcwd()
```

```
'/content/drive/MyDrive/XRA'
```

```
!pip install -r requirements.txt
```

```
Collecting matplotlib==3.10.3 (from -r requirements.txt (line 2))
  Using cached matplotlib-3.10.3-cp312-cp312-manylinux_2_17_x86_64.manyli
ERROR: Could not find a version that satisfies the requirement opencv==4.
ERROR: No matching distribution found for opencv==4.12.0
```

```
!pip install monai
!pip install torchmetrics
```

```
Requirement already satisfied: monai in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: numpy<3.0,>=1.24 in /usr/local/lib/python3
Requirement already satisfied: torch>=2.4.1 in /usr/local/lib/python3.12/
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/li
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/di
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/lo
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/lo
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/
```

```
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/loca
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/loc
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/loc
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/loca
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/li
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/loc
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pytho
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
Requirement already satisfied: torchmetrics in /usr/local/lib/python3.12/
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.12/
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.1
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/l
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/di
Requirement already satisfied: typing_extensions in /usr/local/lib/python
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/lo
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/lo
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/loca
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/loc
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/loc
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/loca
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/li
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/loc
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pytho
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
```

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12
Requirement already satisfied: numpy<2.3.0,>=2 in /usr/local/lib/python3.
```

import packages & dependencies

```
import os

import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import DataLoader, Dataset
from torchvision.io import read_image
from torchvision import transforms
from torch.utils.data import DataLoader
from torchvision.transforms import InterpolationMode

import torch
import torch.nn as nn
from transformers import ViTModel, AutoConfig, ViTImageProcessor, ViTConfig

from transformers import Trainer, TrainingArguments, AutoModelForSemanticSegmentatio
from transformers import EarlyStoppingCallback


from safetensors.torch import load_file

import monai.losses as ml               # monai loss functions library
import monai.metrics as mm              # monai metrics library
from monai.networks.utils import one_hot

from torchmetrics.segmentation import DiceScore, HausdorffDistance

from monai.losses import FocalLoss, TverskyLoss

from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score,

from skimage.filters import sato
import cv2

from transformers.models.vit.modeling_vit import ViTAttention, ViTSelfAttention
```

Image Data & Processing

```
# loading data
# ABS_PATH = '/Users/daofeng/Desktop/______/INM363/CODE/syntax'               # ABS
ABS_PATH = '/content/drive/MyDrive/XRA/syntax'

# define custom data class
class Syntax():
    def __init__(self, root_path, dataset):

        self.root_path =  root_path

        if dataset == 'train':
```

```
            self.images = sorted([root_path + '/train/images/' + i for i in os.listd
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.masks  = sorted([root_path + '/train/masks/' + i for i in os.listdi
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.labels = sorted(os.listdir(root_path + '/train/images/'), key = lam
            # self.annots = root_path + '/train/annotations/' + 'train.json'

        elif dataset == 'val':

            self.images = sorted([root_path + '/val/images/' + i for i in os.listdir
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.masks  = sorted([root_path + '/val/masks/' + i for i in os.listdir(
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.labels = sorted(os.listdir(root_path + '/val/images/'), key = lambd

        elif dataset == 'test':

            self.images = sorted([root_path + '/test/images/' + i for i in os.listdi
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.masks  = sorted([root_path + '/test/masks/' + i for i in os.listdir
                                 key = lambda x: int(os.path.splitext(os.path.basena

            self.labels = sorted(os.listdir(root_path + '/test/images/'), key = lamb

        else:
            raise ValueError("dataset parameter needs to be 'train', 'val', or 'test
            # pass


        self.transform = transforms.Compose([
            transforms.Resize((224,224)),                                  # origina
            transforms.Grayscale(num_output_channels = 1),                 # convert
            transforms.ConvertImageDtype(torch.float32)                    # convert
        ])

        self.ch3_transform = transforms.Compose([                          # second
            transforms.Lambda(lambda x: x.repeat(3, 1, 1))                 # this co
        ])

        self.msk_transform = transforms.Compose([
            transforms.Resize((224,224),
                              interpolation = InterpolationMode.NEAREST),
            transforms.ConvertImageDtype(torch.float32),                    # conver
            transforms.Lambda(lambda pixel: (pixel > 0).float())            # binari
        ])


    def __len__(self):
```

```
        return len(self.labels)

    def __getitem__(self, idx):

        img = read_image(self.images[idx])                  # loads images
        img = self.transform(img)                           # applies transforms
        img = self.ch3_transform(img)                       # converts to 3 channels



        msk = read_image(self.masks[idx])                   # loads masks
        msk = self.transform(msk)                           # applies transforms

        lbl = self.labels[idx]

        return img, msk, lbl
```

```
# initialize datasets
data_train = Syntax(root_path = ABS_PATH, dataset = 'train')
data_val = Syntax(root_path = ABS_PATH, dataset = 'val')
data_test = Syntax(root_path = ABS_PATH, dataset = 'test')
```

```
# define data loaders
BATCHN = [10, 20, 25, 50, 100]                       # different batch sizes batch s

train_loader = DataLoader(dataset = data_train, shuffle = False, batch_size = BATCHN
test_loader = DataLoader(dataset = data_test, shuffle = False, batch_size = BATCHN[3
val_loader = DataLoader(dataset = data_val, shuffle = False, batch_size = BATCHN[4])

train_imgs, train_msks, train_lbls = next(iter(train_loader))
val_imgs, val_msks, val_lbls = next(iter(val_loader))
```

```
# load full train/val/test data from dataloaders
data_train
data_val
data_test

def load_all(loader):
    imgs, msks, lbls = [], [], []

    for i, m, l in loader:
        imgs.append(i)
        msks.append(m)
        lbls.append(l)

    imgs = torch.cat(imgs, dim = 0)             # concatenate batches
    msks = torch.cat(msks, dim = 0)

    return imgs, msks, lbls


train_imgs, train_msks, train_lbls = load_all(train_loader)
val_imgs, val_msks, val_lbls = load_all(val_loader)
test_imgs, test_msks, test_lbls = load_all(test_loader)
```

```
# data augmentations (applied only to train)
import torchvision.transforms.v2 as v2

class AUG(Dataset):
    def __init__(self, imgs: torch.Tensor, msks: torch.Tensor):

        self.x = imgs           # [1000, 3, 512, 512]
        self.y = msks           # [1000, 1, 512, 512]


        self.aug = v2.Compose([
            v2.RandomHorizontalFlip(p = 0.2),
            v2.RandomVerticalFlip(p = 0.2),
            v2.RandomRotation(degrees= [-15, 15], interpolation = InterpolationMode.
            v2.RandomResizedCrop(size = (224, 224), scale = (1.0, 1.4), ratio = (1.0
                                #  interpolation = InterpolationMode.NEAREST,
                                 antialias = True)
        ])

    def __len__(self):
        return self.y.shape[0]

    def __getitem__(self, idx):

        img = self.x[idx]
        msk = self.y[idx]

        # # random geometric augmentations on both x and y
        # aug_xy = self.aug(self.xy)
        # aug_xy = (lambda imgs, msks: imgs, msks,= self.aug() for imgs, msks in sel

            # random geometric augmentations on both x and y
            # random photometric augmentations on only x
            # gamma transform


        aug = self.aug({'images': img, 'masks': msk})
        aug_x = aug['images']
        aug_y = aug['masks']

        print(aug_x.dtype)
        print(aug_y.dtype)
        print(aug_x.shape)
        print(aug_y.shape)


        # ds_aug = {
        #     'train_imgs': aug_x ,
        #     'train_msks': aug_y
        # }

        return aug_x, aug_y
```

```
aug = AUG(train_imgs, train_msks)
```

```
aug_x = aug.x
aug_y = aug.y
torch.unique(aug_y)
```

```
tensor([0.0000, 0.0039, 0.0078, 0.0118, 0.0157, 0.0196, 0.0235, 0.0275,
0.0314,
        0.0353, 0.0392, 0.0431, 0.0471, 0.0510, 0.0549, 0.0588, 0.0627,
0.0667,
        0.0706, 0.0745, 0.0784, 0.0824, 0.0863, 0.0902, 0.0941, 0.0980,
0.1020,
        0.1059, 0.1098, 0.1137, 0.1176, 0.1216, 0.1255, 0.1294, 0.1333,
0.1373,
        0.1412, 0.1451, 0.1490, 0.1529, 0.1569, 0.1608, 0.1647, 0.1686,
0.1725,
        0.1765, 0.1804, 0.1843, 0.1882, 0.1922, 0.1961, 0.2000, 0.2039,
0.2078,
        0.2118, 0.2157, 0.2196, 0.2235, 0.2275, 0.2314, 0.2353, 0.2392,
0.2431,
        0.2471, 0.2510, 0.2549, 0.2588, 0.2627, 0.2667, 0.2706, 0.2745,
0.2784,
        0.2824, 0.2863, 0.2902, 0.2941, 0.2980, 0.3020, 0.3059, 0.3098,
0.3137,
        0.3176, 0.3216, 0.3255, 0.3294, 0.3333, 0.3373, 0.3412, 0.3451,
0.3490,
        0.3529, 0.3569, 0.3608, 0.3647, 0.3686, 0.3725, 0.3765, 0.3804,
0.3843,
        0.3882, 0.3922, 0.3961, 0.4000, 0.4039, 0.4078, 0.4118, 0.4157,
0.4196,
        0.4235, 0.4275, 0.4314, 0.4353, 0.4392, 0.4431, 0.4471, 0.4510,
0.4549,
        0.4588, 0.4627, 0.4667, 0.4706, 0.4745, 0.4784, 0.4824, 0.4863,
0.4902,
        0.4941, 0.4980, 0.5020, 0.5059, 0.5098, 0.5137, 0.5176, 0.5216,
0.5255,
        0.5294, 0.5333, 0.5373, 0.5412, 0.5451, 0.5490, 0.5529, 0.5569,
0.5608,
        0.5647, 0.5686, 0.5725, 0.5765, 0.5804, 0.5843, 0.5882, 0.5922,
0.5961,
        0.6000, 0.6039, 0.6078, 0.6118, 0.6157, 0.6196, 0.6235, 0.6275,
0.6314,
        0.6353, 0.6392, 0.6431, 0.6471, 0.6510, 0.6549, 0.6588, 0.6627,
0.6667,
        0.6706, 0.6745, 0.6784, 0.6824, 0.6863, 0.6902, 0.6941, 0.6980,
0.7020,
        0.7059, 0.7098, 0.7137, 0.7176, 0.7216, 0.7255, 0.7294, 0.7333,
0.7373,
        0.7412, 0.7451, 0.7490, 0.7529, 0.7569, 0.7608, 0.7647, 0.7686,
0.7725,
        0.7765, 0.7804, 0.7843, 0.7882, 0.7922, 0.7961, 0.8000, 0.8039,
0.8078,
```

```
         0.8118, 0.8157, 0.8196, 0.8235, 0.8275, 0.8314, 0.8353, 0.8392,
0.8431,
         0.8471, 0.8510, 0.8549, 0.8588, 0.8627, 0.8667, 0.8706, 0.8745,
0.8784,
         0.8824, 0.8863, 0.8902, 0.8941, 0.8980, 0.9020, 0.9059, 0.9098,
0.9137,
         0.9176, 0.9216, 0.9255, 0.9294, 0.9333, 0.9373, 0.9412, 0.9451,
0.9490,
         0.9529, 0.9569, 0.9608, 0.9647, 0.9686, 0.9725, 0.9765, 0.9804,
0.9843,
         0.9882, 0.9922, 0.9961, 1.0000])
```

```
aug_y = (aug_y > 0.0).to(torch.float32)
```

```
torch.unique(aug_y)
```

```
tensor([0., 1.])
```

```
# define sato + sobel filter function

def apply_filters(img_ds):

    output = []

    for img in img_ds:
        grayscale_ch = img[0].numpy()               # grayscale channel to np array

        # # clahe channel
        # clahe_ch = cv2.createCLAHE(clipLimit = 2, tileGridSize = (8, 8))

        # sato filter
        sato_ch = sato(grayscale_ch, sigmas=(1,2,3), black_ridges=True)     # black_
        sato_ch = np.clip(sato_ch, 0.0, 1.0).astype(np.float32)             # min-ma

        # sobel filter
        grad_x = cv2.Sobel(grayscale_ch, cv2.CV_32F, 1, 0, ksize = 3)       # kernel
        grad_y = cv2.Sobel(grayscale_ch, cv2.CV_32F, 0, 1, ksize = 3)       # CV_32F

        grad_m = cv2.magnitude(grad_x, grad_y)                              # gradie
        grad_m = grad_m / (grad_m.max() + 1e-6)                             # normal
        sobel_ch = grad_m.astype(np.float32)

        stacked_ch = np.stack([grayscale_ch, sato_ch, sobel_ch], axis = 0)  # axis 0

        output.append(torch.from_numpy(stacked_ch))                         # append

    return torch.stack(output)
```

```
# apply fiters
train_imgs = apply_filters(train_imgs)
val_imgs = apply_filters(val_imgs)
test_imgs = apply_filters(test_imgs)
aug_imgs = apply_filters(aug_x)
```

```
# compute normalizations
# use train img mean & std              train_imgs[:, 0, :, :].std().item()

grayscale_mean = [aug_imgs[:, 0, :, :].mean().item(),
                  aug_imgs[:, 0, :, :].mean().item(),
                  aug_imgs[:, 0, :, :].mean().item()]

grayscale_std = [aug_imgs[:, 0, :, :].std().item(),
                 aug_imgs[:, 0, :, :].std().item(),
                 aug_imgs[:, 0, :, :].std().item()]

# channel wise mean
channel_mean = [aug_imgs[:, 0, :, :].mean().item(),
                aug_imgs[:, 1, :, :].mean().item(),
                aug_imgs[:, 2, :, :].mean().item()]

channel_std = [aug_imgs[:, 0, :, :].std().item(),
               aug_imgs[:, 1, :, :].std().item(),
               aug_imgs[:, 2, :, :].std().item()]
```

Start coding or generate with AI.

```
# # data augmentations (applied only to train)
# import torchvision.transforms.v2 as v2

# class AUG(Dataset):
#     def __init__(self, ds: torch.Tensor):

#         self.xy = ds
#         self.x = ds.x           # [1000, 3, 512, 512]
#         self.y = ds.y           # [1000, 1, 512, 512]


#         self.aug = v2.Compose([
#             v2.RandomHorizontalFlip(p = 0.2),
#             v2.RandomVerticalFlip(p = 0.2),
#             v2.RandomRotation(degrees= [-15, 15], interpolation = InterpolationMod
#             v2.RandomResizedCrop(size = (224, 224), scale = (1.0, 1.4), ratio = (1
#                                 #  interpolation = InterpolationMode.NEAREST,
#                                  antialias = True)
#         ])
```

```
#     def __len__(self):
#         return self.y.shape[0]

#     def __getitem__(self, idx):

#         img = self.x[idx]
#         msk = self.y[idx]

#         # # random geometric augmentations on both x and y
#         # aug_xy = self.aug(self.xy)
#         # aug_xy = (lambda imgs, msks: imgs, msks,= self.aug() for imgs, msks in s

#             # random geometric augmentations on both x and y
#             # random photometric augmentations on only x
#             # gamma transform


#         aug = self.aug({'images': img, 'masks': msk})
#         aug_x = aug['images']
#         aug_y = aug['masks']

#         print(aug_x.dtype)
#         print(aug_y.dtype)
#         print(aug_x.shape)
#         print(aug_y.shape)


#         # ds_aug = {
#         #     'train_imgs': aug_x ,
#         #     'train_msks': aug_y
#         # }

#         return aug_x, aug_y
```

```
# use processor and collator to prepare the images
processor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224')
# model_input = processor(images = train_imgs, return_tensors = 'pt')

# processor settings
# processor.size = {'height': 512, 'width': 512}
processor.size = {'height': 224, 'width': 224}
processor.do_convert_rgb = False        # grayscale / custom channels
processor.do_rescale = True


# processor.image_mean = grayscale_mean           # defaults to [0.5, 0.5, 0.5]
# processor.image_std = grayscale_std
# processor.do_normalize = True

processor.image_mean = channel_mean           # defaults to [0.5, 0.5, 0.5]
processor.image_std = channel_std
processor.do_normalize = True
```

Fetching 1 files: 100% 1/1 [00:00<00:00, 123.78it/s]

```
# processing each dataset for trainer
# train_x = processor(images = train_imgs, return_tensors = 'pt')
# train_x = train_x['pixel_values']
# train_y = train_msks

train_x = processor(images = aug_imgs, return_tensors = 'pt')
train_x = train_x['pixel_values']
train_y = aug_y

val_x = processor(images = val_imgs, return_tensors = 'pt')
val_x = val_x['pixel_values']
val_y = val_msks

test_x = processor(images = test_imgs, return_tensors = 'pt')
test_x = test_x['pixel_values']
test_y = test_msks
```

```
# collate into one dataset dict of X, y
class SYN(Dataset):
    def __init__(self, pixel_values: torch.Tensor, masks: torch.Tensor):
        self.x = pixel_values
        self.y = masks

    def __len__(self):
        return self.y.size(0)

    def __getitem__(self, idx):
        dat = {
            'pixel_values': self.x[idx],
            'labels': self.y[idx]
        }
        return dat

ds_train = SYN(train_x, train_y)
ds_val = SYN(val_x, val_y)
ds_test = SYN(test_x, test_y)
```

Defining Model

```
model_id = 'google/vit-base-patch16-224'
# model_id = 'google/vit-base-patch32-224-in21k'
config = ViTConfig.from_pretrained(model_id)
VTmodel = ViTModel.from_pretrained(model_id)
```

```
Some weights of ViTModel were not initialized from the model checkpoint a
You should probably TRAIN this model on a down-stream task to be able to
```

```
# config = ViTConfig(
#     hidden_size = 768,
#     num_hidden_layers = 12,
#     num_attention_heads = 12,
#     image_size = 512,
#     patch_size = 32,
#     hidden_dropout_prob = 0.0,
#     attention_probs_dropout_prob = 0.0,
#     qkv_bias = True
# )

config = ViTConfig(
    hidden_size = 768,
    num_hidden_layers = 12,
    num_attention_heads = 12,
    image_size = 224,
    patch_size = 16,
    hidden_dropout_prob = 0.0,
    attention_probs_dropout_prob = 0.0,
    qkv_bias = True
)
```

```
class SNR_ATTN(ViTSelfAttention):
    def __init__(self, config):
        super().__init__(config)

        self.patch_size = 16

        # snr params
        self.snr_bias_w = nn.Parameter(torch.tensor(1.0))           # initialize to
        self.snr_scaler = nn.Parameter(torch.tensor(1.0))           # initialize to


    def transpose_for_scores(self, x):

        B, N, C = x.size()

        dim_t = x.size()[:-1] + (self.num_attention_heads, self.attention_head_size)

        x = x.view(dim_t)
        x = x.permute(0, 2, 1, 3)       # [B, N, n_attn_heads, head_size]

        return x


    def forward(self, hidden_states, pixel_values = None, snr_values = None):

        B, N, C, = hidden_states.size()
        A = self.num_attention_heads                             # number of attent
        # D = int(768 / 12)
        D = self.attention_head_size
```

```
        mixed_query = self.query(hidden_states)

        query = self.transpose_for_scores(self.query(hidden_states))
        key = self.transpose_for_scores(self.key(hidden_states))
        value = self.transpose_for_scores(self.value(hidden_states))

        # compute attention
        attn_scores = torch.matmul(query, key.transpose(-1, -2))
        attn_scores = attn_scores / torch.sqrt(torch.tensor(D, dtype = torch.float32

        # get snr values from parent class
        if snr_values is None:
            snr_values = getattr(self, '_snr_values', None)

        if snr_values is not None:

            cls_tokens = torch.zeros(B, 1, device = snr_values.device, dtype = snr_v
            cls_snr_map = torch.cat([cls_tokens, snr_values], dim = 1)


            bias = (self.snr_bias_w * cls_snr_map / self.snr_scaler).unsqueeze(-1)
            bias = bias.expand(-1, -1, A).permute(0, 2, 1)
            bias_ij = bias.unsqueeze(-1) + bias.unsqueeze(-2)
            snr_attn_scores = attn_scores + bias_ij

        else:
            pass


        attn = snr_attn_scores.softmax(dim = -1)
        attention_qkvs = torch.matmul(attn, value)

        attention_qkvs = attention_qkvs.permute(0, 2, 1, 3).contiguous()
        attn_dim = attention_qkvs.size()[: -2] + (self.all_head_size,)
        attention_qkvs = attention_qkvs.view(attn_dim)

        # out = self.output(attention_qkvs, hidden_states)


        return (attention_qkvs, attn)
```

```
class ViSNR(nn.Module):
    def __init__(self, model_id, img_size, patch_size, freeze):
        super().__init__()

        # loading the pre-trained model
        # self.config = ViTConfig.from_pretrained(model_id)

        self.config = ViTConfig(
            hidden_size = 768,
            num_hidden_layers = 12,
```

```
            num_attention_heads = 12,
            image_size = 224,
            patch_size = 16,
            hidden_dropout_prob = 0.0,
            attention_probs_dropout_prob = 0.0,
            qkv_bias = True
            )


        # self.vit = ViTModel(self.config)
        self.vit = ViTModel.from_pretrained(model_id, config = self.config)

        # self.embeddings = self.vit.embeddings
        # self.vit.get_position_embeddings



        # define params
        self.img_size = 224                                 # default ViT input size,
        self.patch_size = 16                                # default 16 patches
        self.num_patches = img_size // patch_size           # number of patches in ima
        self.grid_size = (img_size // patch_size) ** 2      # grid size of each patch

        # backbone vit encoder param freeze
        if freeze:
            for par in self.vit.parameters():
                par.requires_grad = False                   # ViT params/weights froze
            print('encoder backbone frozen')
        else:
            print('encoder backbone training enabled')

        # replace attention layers

        for block in self.vit.encoder.layer:
            block.attention.attention = SNR_ATTN(self.config)


        # define decoder
        self.decoder = nn.Sequential(
            nn.LayerNorm(768),                                # input dim [B, grid_siz
            nn.Linear(768, 1024),                             # fc layer to [B, 196, 1
            nn.GELU(),                                        # gelu activation
            nn.Dropout(0.1),
            nn.Linear(1024, 512),                             # [B, 196, 512]
            nn.GELU(),                                        # gelu activation
            nn.Dropout(0.1),
            nn.Linear(512, 64),                               # [B, 196, 64]
        )


        # define spatial upsampling
        self.upsample = nn.Sequential(
            # nn.convTranspose2d(in_channels, out_channels, kernel_size, stride, paddi
            # dimensions [B, C, H, W]
```

```
            nn.ConvTranspose2d(64, 128, kernel_size = 4, stride = 2, padding = 1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size = 4, stride = 2, padding = 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, kernel_size = 4, stride = 2, padding = 1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size = 4, stride = 2, padding = 1),
            nn.BatchNorm2d(16),
            nn.ReLU(),

            nn.Conv2d(16, 1, kernel_size = 1),
            )


        # define loss functions

        # bce + dice


        # self.loss_bce = nn.BCEWithLogitsLoss(pos_weight = torch.tensor((1 - 0.0196)
        # # self.loss_dice = ml.dice.DiceLoss(sigmoid = True, squared_pred = True, rec
        # self.loss_dice = ml.dice.DiceLoss(sigmoid = True, reduction = 'mean', smooth

        # self.a = 0.4

        # self.combined_loss = lambda y_pred, y_true: self.a * self.loss_bce(y_pred, y

        # # combined weighted bce + dice loss
        # self.loss_function = self.combined_loss



        # # Focal Loss
        # self.loss_focal = FocalLoss(gamma = 2, alpha = 0.90, weight = None,
        #                             reduction = 'mean')

        # self.loss_tversky = TverskyLoss(alpha = 0.6, beta = 0.4, reduction = 'mean',
        #                                 sigmoid = True, smooth_nr = 1e-4, smooth_dr

        # self.a = 0.6

        # self.combined_loss = lambda y_pred, y_true: self.a * self.loss_focal(y_pred,

        # # combined weighted focal + tversky loss
        # self.loss_function = self.combined_loss

        # Focal Loss
        # PARAMS focal[gamma = 2, alpha = 0.90] ; Tversky[0.3, 0.7] self.a = 0.6

        self.loss_focal = FocalLoss(gamma = 2, alpha = 0.90, weight = None,
                                    reduction = 'mean')
```

```
        self.loss_tversky = TverskyLoss(alpha = 0.6, beta = 0.4, reduction = 'mean',
                                        sigmoid = True, smooth_nr = 1e-4, smooth_dr =

        self.a = 0.4

        self.combined_loss = lambda y_pred, y_true: self.a * self.loss_focal(y_pred, y

        # combined weighted focal + tversky loss
        self.loss_function = self.combined_loss



    def compute_patch_snr(self, pixel_values, patch_size = 16, epsilon = 1e-8):

        B, C, H, W = pixel_values.shape            # assign shape variables

        # reshaping patches to dim [B, C, grid, grid, p, p]
        patches = pixel_values.unfold(2, patch_size, patch_size)      # [B, C, grid_H,
        patches = patches.unfold(3, patch_size, patch_size)           # [B, C, grid_H,
        # reshaping patches to dim [B, C, N, p, p] 5D tensor
        patches = patches.contiguous().view(B, C, -1, patch_size, patch_size)   # [B,

        # compute mean and std dev per patch over channels and pxp

        Ps = patches.mean(dim = [1, 3, 4], keepdim = False)              # [B, N]
        Pn = patches.std(dim = [1, 3, 4], keepdim = False)               # [B, N]

        snr = Ps / (Pn + epsilon)                                       # [B, N]

        return snr

    def send_to_attn_layer(self, snr_vals):

        for block in self.vit.encoder.layer:
            if hasattr(block.attention, 'attention') and isinstance(block.attention.at
                block.attention.attention._snr_values = snr_vals



    def forward(self, pixel_values, labels = None):

        snr_values = self.compute_patch_snr(pixel_values, self.patch_size)
        self.send_to_attn_layer(snr_values)


        encoder_outputs = self.vit(pixel_values, return_dict = True)        # hidden s
        patch_embeddings = encoder_outputs.last_hidden_state[:, 1:, :]      # [batch,
        patch_features = self.decoder(patch_embeddings)                     # passes e

        batch_size = patch_features.shape[0]                                # returns

        spatial_logits = patch_features.transpose(1, 2).reshape(
            batch_size, 64, self.num_patches, self.num_patches)              # transpo
```

```
        ups_logits = self.upsample(spatial_logits)                          # upsample


        if labels is not None:                                              # if groun
            labels = (labels > 0.5).float()
            loss = self.loss_function(ups_logits, labels)
            return {'loss': loss, "logits": ups_logits}                     # return l

        else:
            return ups_logits                                               # return o


    def predict(self, pixel_values, threshold):
        with torch.no_grad():
            logits = self.forward(pixel_values)                             # computes
            probas = torch.sigmoid(logits)                                  # computes
            bin_msk = (probas > threshold).float()                          # creates
        return bin_msk                                                      # bin_msk

    def unfreeze(self):
        for par in self.vit.parameters():
            par.requires_grad = True                                         # unfreez
        print('vit encoder unfrozen')
```

Training

```
dice_mm = mm.DiceMetric(include_background=False, reduction = 'mean', get_not_nans=F
```

```python
# define eval metrics

def eval_metrics(evalpred):
    logits = evalpred.predictions                           # returns model pred on
    y_true = evalpred.label_ids                             # returns gt mask on val

    probas = 1 / (1 + np.exp(-logits))                      # sigmoid function
    y_pred = (probas > 0.2).astype(np.float32)              # convert to binary

    y_pred_fl = y_pred.ravel().astype(int)                  # flatten for sklearn
    y_true_fl = y_true.ravel().astype(int)

    y_pred_tensor = torch.from_numpy(y_pred.astype(np.float32))
    y_true_tensor = torch.from_numpy(y_true.astype(np.float32))

    y_pred_1hot = one_hot(y_pred_tensor, num_classes = 2)
    y_true_1hot = one_hot(y_true_tensor, num_classes = 2)

    # compute metrics
    acc = accuracy_score(y_pred_fl, y_true_fl)
    f1 = f1_score(y_pred_fl, y_true_fl, zero_division = 0)
    prec = precision_score(y_pred_fl, y_true_fl, zero_division = 0)
    rec = recall_score(y_pred_fl, y_true_fl, zero_division = 0)
    js = jaccard_score(y_pred_fl, y_true_fl, zero_division = 0)

    # dice and iou scores
    # dice_score = DiceScore(num_classes = 2, include_background = False)
    # dice = dice_score(y_pred_tensor, y_true_tensor)

    dice_score = dice_mm(y_pred_1hot, y_true_1hot)
    dice = dice_mm.aggregate().item()
    dice_mm.reset()                                        # reset dice for next epoc


    # hausdorff = HausdorffDistance(num_classes = 2, include_background = False,
    #                        distance_metric = 'euclidean', directed = False)

    # hd = hausdorff(y_pred_tensor, y_true_tensor)


    res_dict = {'acc': acc, 'dice': dice, 'f1': f1, 'rec': rec, 'prec': prec, 'jacc'

    return res_dict
```

```python
visnr = ViSNR(model_id, img_size = 224, patch_size = 16, freeze = True)
```

```
Some weights of ViTModel were not initialized from the model checkpoint a
You should probably TRAIN this model on a down-stream task to be able to
encoder backbone frozen
```

```
# send to gpu
torch.backends.mps.is_built()                       # check that mps build is compli

if torch.backends.mps.is_available():
    device = torch.device('mps')
else:
    device = torch.device('cpu')

print(device)

visnr.to(device)
```

```
cpu
ViSNR(
  (vit): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16,
16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): SNR_ATTN(
              (query): Linear(in_features=768, out_features=768,
bias=True)
              (key): Linear(in_features=768, out_features=768,
bias=True)
              (value): Linear(in_features=768, out_features=768,
bias=True)
            )
            (output): ViTSelfOutput(
              (dense): Linear(in_features=768, out_features=768,
bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): Linear(in_features=768, out_features=3072,
bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): Linear(in_features=3072, out_features=768,
bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          (layernorm_after): LayerNorm((768,), eps=1e-12,
```

```
      elementwise_affine=True)
              )
            )
          )
          (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (pooler): ViTPooler(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (activation): Tanh()
          )
        )
        (decoder): Sequential(
          (0): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (1): Linear(in_features=768, out_features=1024, bias=True)
          (2): GELU(approximate='none')
          (3): Dropout(p=0.1, inplace=False)
          (4): Linear(in_features=1024, out_features=512, bias=True)
          (5): GELU(approximate='none')
          (6): Dropout(p=0.1, inplace=False)
```

```
targs1 = TrainingArguments(
    output_dir = 'training/visnr_aug/p1',                        # separate training ou
    # logging_dir = 'training/vit-b16/logs',
    report_to = ['none'],
    num_train_epochs = 15,                                   # training decoder, use mor
    per_device_train_batch_size = 25,
    per_device_eval_batch_size = 25,

    learning_rate = 1e-4 ,                                   # try 5e-4 for bigger steps
    weight_decay = 0.01 ,

    warmup_ratio = 0.10,
    lr_scheduler_type='cosine',

    # using fp32 -- full precision
    fp16 = False,                                           # disable half precision
    bf16 = False,                                           # disable bfloat precision

    logging_strategy = 'epoch',
    save_strategy = 'epoch',
    eval_strategy = 'epoch',
    load_best_model_at_end = True,                          # trainer saves model
    metric_for_best_model = 'eval_loss',                    # ['eval_loss', 'eval_runtim
    greater_is_better = False,                               # false for BCE + dice loss

    save_total_limit = 3,
    save_safetensors = True,
)
```

```
# instantiate trainer
trainer1 = Trainer(
    model = visnr,
    args = targs1,
    train_dataset = ds_train,
    eval_dataset = ds_val,
    compute_metrics = eval_metrics,
    callbacks = [EarlyStoppingCallback(early_stopping_patience = 5)]
)
```

```
from accelerate.state import AcceleratorState
from accelerate import Accelerator


AcceleratorState._reset_state()
accel = Accelerator()
```

```
trainer1.train()
```

[600/600 02:53, Epoch 15/15]

Epoch	Training Loss	Validation Loss	Acc	Dice	F1	Rec	Prec	J.
1	0.562900	0.571720	0.021255	0.041451	0.041625	0.021255	1.000000	0.
2	0.552700	0.569432	0.026087	0.041644	0.041819	0.021356	0.999920	0.
3	0.549300	0.558817	0.125727	0.045994	0.046207	0.023652	0.996362	0.
4	0.547600	0.559274	0.183155	0.049051	0.049292	0.025271	0.996292	0.
5	0.545800	0.561487	0.197923	0.049948	0.050198	0.025747	0.997210	0.
6	0.544900	0.556396	0.374427	0.061493	0.061869	0.031953	0.970520	0.
7	0.543900	0.560467	0.296518	0.056153	0.056466	0.029062	0.990375	0.
8	0.543300	0.558802	0.290665	0.055771	0.056077	0.028855	0.991331	0.
9	0.542900	0.555711	0.385727	0.062908	0.063298	0.032709	0.976488	0.
10	0.542500	0.555573	0.386597	0.063075	0.063468	0.032798	0.977899	0.
11	0.542200	0.554000	0.417115	0.065651	0.066075	0.034202	0.970121	0.
12	0.541800	0.554930	0.395662	0.063811	0.064211	0.033198	0.975494	0.
13	0.541700	0.554686	0.403008	0.064437	0.064845	0.033539	0.973820	0.
14	0.541600	0.555088	0.393254	0.063623	0.064021	0.033096	0.976286	0.
15	0.541600	0.554836	0.400646	0.064256	0.064661	0.033440	0.974711	0

```
TrainOutput(global_step=600, training_loss=0.5456506411234537, metrics=
```

```
# reinstantiate
visnrp2 = ViSNR(model_id, img_size = 224, patch_size = 16, freeze = False)

# load params
checkpoint = trainer1.state.best_model_checkpoint
# checkpoint = 'training/vit16/phase_1/checkpoint-2760'

# load safetensors file
w_path = checkpoint + '/model.safetensors'

print(w_path)
# load weights into model
state = load_file(w_path, device = 'cpu')
visnrp2.load_state_dict(state)
```

```
print(device)
visnrp2.to(device)
```

```
      (encoder): ViTEncoder(
        (layer): ModuleList(
          (0-11): 12 x ViTLayer(
            (attention): ViTAttention(
              (attention): SNR_ATTN(
                (query): Linear(in_features=768, out_features=768,
bias=True)
                (key): Linear(in_features=768, out_features=768,
bias=True)
                (value): Linear(in_features=768, out_features=768,
bias=True)
              )
              (output): ViTSelfOutput(
                (dense): Linear(in_features=768, out_features=768,
bias=True)
                (dropout): Dropout(p=0.0, inplace=False)
              )
            )
            (intermediate): ViTIntermediate(
              (dense): Linear(in_features=768, out_features=3072,
bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): ViTOutput(
              (dense): Linear(in_features=3072, out_features=768,
bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (layernorm_before): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
            (layernorm_after): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
          )
        )
      )
      (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (pooler): ViTPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (decoder): Sequential(
      (0): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (1): Linear(in_features=768, out_features=1024, bias=True)
      (2): GELU(approximate='none')
      (3): Dropout(p=0.1, inplace=False)
      (4): Linear(in_features=1024, out_features=512, bias=True)
      (5): GELU(approximate='none')
      (6): Dropout(p=0.1, inplace=False)
      (7): Linear(in_features=512, out_features=64, bias=True)
```

```
    )
    (upsample): Sequential(
      (0): ConvTranspose2d(64, 128, kernel_size=(4, 4), stride=(2, 2),
padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU()
      (3): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2),
padding=(1, 1))
```

```
targs2 = TrainingArguments(
    output_dir = 'training/visnr_aug/p2',                     # separate training ou
    # logging_dir = 'training/vit-b16/logs',
    report_to = ['none'],
    num_train_epochs = 135,                                # training decoder, use mo
    per_device_train_batch_size = 25,
    per_device_eval_batch_size = 25,

    learning_rate = 1e-4 ,                                # try 5e-4 for bigger steps
    weight_decay = 0.01 ,

    warmup_ratio = 0.10,
    lr_scheduler_type='cosine',

    # using fp32 -- full precision
    fp16 = False,                                        # disable half precision
    bf16 = False,                                        # disable bfloat precision

    logging_strategy = 'epoch',
    save_strategy = 'epoch',
    eval_strategy = 'epoch',
    load_best_model_at_end = True,                       # trainer saves model
    metric_for_best_model = 'eval_loss',                 # ['eval_loss', 'eval_runtim
    greater_is_better = False,                            # false for BCE + dice loss

    save_total_limit = 3,
    save_safetensors = True,
)
```

```
trainer2 = Trainer(
    model = visnrp2,
    args = targs2,
    train_dataset = ds_train,
    eval_dataset = ds_val,
    compute_metrics = eval_metrics,
    callbacks = [EarlyStoppingCallback(early_stopping_patience = 30)]
)
```

```
trainer2.train()
```

[5400/5400 35:30, Epoch 135/135]

Epoch	Training Loss	Validation Loss	Acc	Dice	F1	Rec	Prec
1	0.542000	0.554675	0.397792	0.064019	0.064422	0.033311	0.975475
2	0.541900	0.555242	0.434108	0.067166	0.067610	0.035032	0.965297
3	0.541900	0.555001	0.422651	0.066131	0.066558	0.034464	0.968433
4	0.541800	0.554692	0.407573	0.064810	0.065226	0.033745	0.972447
5	0.541100	0.563452	0.688293	0.092700	0.093685	0.049928	0.757969
6	0.535000	0.547947	0.399091	0.066344	0.065012	0.033618	0.982902
7	0.521200	0.531855	0.461002	0.073030	0.071919	0.037326	0.982569
8	0.514800	0.529340	0.441660	0.072971	0.069950	0.036259	0.987857
9	0.510200	0.529042	0.403523	0.075161	0.065781	0.034023	0.988003
10	0.508000	0.522633	0.434088	0.102604	0.067568	0.035010	0.964687
11	0.504800	0.540195	0.262143	0.077504	0.053619	0.027561	0.983431
12	0.500500	0.530170	0.350838	0.096963	0.059819	0.030860	0.971621
13	0.498600	0.502057	0.631087	0.173994	0.095002	0.050114	0.911011
14	0.495300	0.521418	0.438319	0.106550	0.068559	0.035532	0.972550
15	0.491200	0.509715	0.525539	0.141059	0.077796	0.040574	0.941555
16	0.487400	0.517371	0.425402	0.130743	0.065475	0.033910	0.947036
17	0.484300	0.503754	0.548961	0.166110	0.080286	0.041961	0.926220
18	0.481400	0.510732	0.489826	0.135807	0.073781	0.038371	0.956019
19	0.477800	0.518080	0.392608	0.125463	0.062782	0.032455	0.957144
20	0.475400	0.504238	0.474369	0.153747	0.070541	0.036648	0.938447
21	0.472700	0.516841	0.390798	0.125749	0.062666	0.032392	0.958105
22	0.469500	0.494071	0.563138	0.177039	0.081823	0.042825	0.915816
23	0.467100	0.462945	0.807865	0.242964	0.160439	0.088433	0.863733
24	0.465600	0.521145	0.352063	0.105335	0.060166	0.031040	0.975771
25	0.461400	0.514465	0.367742	0.129358	0.060365	0.031167	0.955512
26	0.458000	0.514917	0.358138	0.130873	0.058606	0.030246	0.940008

27	0.455300	0.460420	0.722543	0.226780	0.119523	0.064084	0.886022
28	0.454300	0.474762	0.647509	0.205745	0.097029	0.051308	0.891029
29	0.450600	0.485249	0.549377	0.181141	0.079774	0.041697	0.918953
30	0.446400	0.465773	0.705732	0.237462	0.110029	0.058794	0.855833
31	0.444400	0.475453	0.623331	0.201232	0.092110	0.048542	0.898976
32	0.442100	0.457435	0.708480	0.238288	0.112057	0.059907	0.865440
33	0.439400	0.445827	0.772557	0.266393	0.135420	0.073661	0.838041
34	0.436200	0.465072	0.668879	0.236334	0.099609	0.052860	0.861727
35	0.432800	0.464365	0.696591	0.247489	0.106402	0.056754	0.849870
36	0.430200	0.468974	0.614356	0.207655	0.090315	0.047541	0.900682
37	0.425500	0.458876	0.664594	0.220887	0.103624	0.054932	0.912126
38	0.420500	0.477445	0.498468	0.195271	0.072701	0.037837	0.924982
39	0.416300	0.422021	0.827925	0.296235	0.172205	0.095909	0.842083
40	0.410000	0.439651	0.724661	0.254755	0.122071	0.065473	0.900607
41	0.407900	0.434071	0.739898	0.263289	0.127934	0.068875	0.897625
42	0.401500	0.396187	0.876271	0.341755	0.219231	0.126595	0.817272
43	0.397700	0.415883	0.781926	0.292836	0.145110	0.079150	0.870775
44	0.394100	0.425276	0.724373	0.291541	0.118291	0.063460	0.869889
45	0.389900	0.408983	0.796662	0.299001	0.155753	0.085414	0.882473
46	0.383100	0.416746	0.752986	0.299868	0.130954	0.070769	0.875614
47	0.379900	0.422349	0.669952	0.282671	0.102151	0.054210	0.883349
48	0.377700	0.423309	0.675125	0.270915	0.105684	0.056126	0.903129
49	0.374900	0.424615	0.681843	0.277591	0.106891	0.056837	0.895764
50	0.376400	0.412024	0.736524	0.303439	0.123764	0.066589	0.875445
51	0.368200	0.386954	0.865147	0.346546	0.212052	0.121060	0.853728
52	0.365200	0.396823	0.812540	0.311547	0.168114	0.092811	0.891179
53	0.361100	0.403612	0.740496	0.311567	0.125927	0.067818	0.879486
54	0.357700	0.377494	0.886450	0.356808	0.242119	0.141072	0.853358
55	0.354300	0.385005	0.828357	0.334749	0.176642	0.098348	0.866256
56	0.354200	0.400066	0.770717	0.299156	0.143649	0.078018	0.904770

57	0.352300	0.371381	0.873424	0.360628	0.221463	0.127385	0.847010
58	0.345700	0.376177	0.840687	0.352787	0.185959	0.104308	0.856129
59	0.340900	0.395759	0.773536	0.330103	0.139460	0.075857	0.863363
60	0.337100	0.365335	0.877975	0.374716	0.227448	0.131407	0.845116
61	0.337000	0.377987	0.793040	0.349663	0.150409	0.082393	0.861914
62	0.331600	0.394694	0.744325	0.325956	0.126368	0.068132	0.869988
63	0.328100	0.373793	0.827368	0.348947	0.175352	0.097584	0.863536
64	0.327300	0.370958	0.825401	0.354461	0.173821	0.096629	0.864141
65	0.325500	0.356480	0.882866	0.390943	0.229441	0.133368	0.820469
66	0.322300	0.357688	0.896114	0.379370	0.255462	0.150685	0.838515
67	0.318100	0.351587	0.914119	0.390207	0.292829	0.177476	0.836564
68	0.316300	0.357141	0.882205	0.378784	0.234709	0.136156	0.849856
69	0.313200	0.351129	0.881687	0.381508	0.232660	0.134930	0.843883
70	0.311800	0.343620	0.925960	0.405369	0.321540	0.199656	0.825453
71	0.307900	0.359585	0.856997	0.378987	0.198485	0.112664	0.833062
72	0.306400	0.343635	0.907541	0.405962	0.274011	0.164451	0.820919
73	0.303900	0.358037	0.858887	0.375189	0.203471	0.115605	0.847976
74	0.301000	0.345742	0.921071	0.400519	0.307717	0.189114	0.825312
75	0.298000	0.349833	0.886981	0.380655	0.240983	0.140555	0.844117
76	0.295400	0.343845	0.896239	0.402325	0.250993	0.148241	0.817942
77	0.295100	0.341662	0.916117	0.398876	0.296443	0.180377	0.831445
78	0.290200	0.340130	0.904454	0.398560	0.270354	0.161369	0.832814
79	0.288200	0.342434	0.911137	0.402824	0.281889	0.170174	0.820582
80	0.286300	0.347986	0.888013	0.388474	0.241253	0.140920	0.837643
81	0.282500	0.346674	0.889197	0.406792	0.235858	0.138184	0.804529
82	0.282300	0.336934	0.918293	0.414788	0.296254	0.181321	0.809142
83	0.280900	0.348534	0.875779	0.388640	0.221739	0.127901	0.832584
84	0.277600	0.345445	0.912314	0.382099	0.292927	0.176759	0.854554
85	0.275200	0.337913	0.918530	0.410835	0.298107	0.182466	0.813985
86	0.272500	0.344540	0.891747	0.396654	0.244437	0.143508	0.823859

86	0.272500	0.344540	0.891747	0.396654	0.244437	0.143508	0.823839
87	0.269300	0.333174	0.938842	0.412890	0.364283	0.233795	0.824417
88	0.267500	0.334774	0.936174	0.421839	0.346773	0.221589	0.797070
89	0.267300	0.337892	0.926322	0.404045	0.324535	0.201539	0.832743
90	0.263500	0.329509	0.942739	0.416940	0.378452	0.245976	0.820179
91	0.261200	0.331319	0.939299	0.423076	0.358378	0.231112	0.797581
92	0.259700	0.330474	0.939766	0.417984	0.365585	0.235517	0.816531
93	0.256700	0.332207	0.932538	0.410270	0.341663	0.215538	0.823610
94	0.254200	0.329397	0.942578	0.414271	0.377809	0.245427	0.820244
95	0.252400	0.328031	0.947467	0.426671	0.392266	0.260084	0.797661
96	0.250500	0.333753	0.918993	0.406162	0.299641	0.183550	0.815298
97	0.249900	0.329159	0.943510	0.417822	0.379782	0.247693	0.813727
98	0.247700	0.329084	0.942093	0.422722	0.370912	0.241136	0.803160
99	0.245900	0.327488	0.939871	0.430893	0.357472	0.231261	0.786948
100	0.244200	0.330755	0.934037	0.414333	0.344119	0.218167	0.814140
101	0.242200	0.328997	0.945635	0.419419	0.387810	0.254918	0.810150
102	0.241100	0.328553	0.947545	0.423782	0.392383	0.260271	0.796863
103	0.239700	0.328116	0.947527	0.423167	0.391678	0.259873	0.794782
104	0.238500	0.329407	0.946903	0.415285	0.394810	0.260517	0.814867
105	0.236800	0.331401	0.946130	0.410547	0.392395	0.258064	0.818402
106	0.235800	0.328173	0.952333	0.427331	0.414421	0.280434	0.793582
107	0.234800	0.326535	0.951046	0.423843	0.408187	0.274670	0.794290
108	0.232100	0.331640	0.946885	0.416205	0.393794	0.259957	0.811679
109	0.231800	0.333755	0.933271	0.415304	0.337703	0.213995	0.800413
110	0.230400	0.330474	0.946669	0.416020	0.390627	0.257962	0.804220
111	0.230400	0.328101	0.947610	0.419201	0.393607	0.261018	0.799962
112	0.229900	0.329278	0.949497	0.417581	0.404306	0.269790	0.806348
113	0.228300	0.329190	0.949235	0.421847	0.401463	0.267856	0.801003
114	0.227400	0.326338	0.950957	0.423436	0.408616	0.274718	0.797149
115	0.227200	0.327084	0.950624	0.421620	0.407269	0.273390	0.798097

116	0.225600	0.329977	0.949404	0.419875	0.401944	0.268405	0.799930
117	0.225500	0.328301	0.950604	0.420533	0.407501	0.273472	0.799184
118	0.225200	0.326747	0.952142	0.424808	0.412966	0.279300	0.791997
119	0.223900	0.328725	0.949802	0.419772	0.404069	0.270219	0.800675
120	0.224000	0.326109	0.950768	0.422720	0.407109	0.273583	0.795232
121	0.223700	0.326458	0.950931	0.422027	0.408400	0.274557	0.796854
122	0.222800	0.327828	0.951305	0.423771	0.409314	0.275753	0.793783
123	0.223000	0.327001	0.950949	0.422605	0.407358	0.274057	0.793141
124	0.222900	0.328160	0.951490	0.423931	0.410159	0.276551	0.793530
125	0.222900	0.328939	0.950381	0.421863	0.404595	0.271557	0.793178
126	0.222200	0.326901	0.951643	0.423160	0.410427	0.276993	0.791908
127	0.222200	0.329392	0.950359	0.420728	0.405269	0.271863	0.795752
128	0.221500	0.326642	0.952133	0.424411	0.412378	0.278983	0.790225
129	0.221700	0.328926	0.950800	0.421583	0.407281	0.273732	0.795288
130	0.221200	0.329544	0.950486	0.421226	0.405699	0.272322	0.795143
131	0.221100	0.327622	0.951100	0.422962	0.408104	0.274733	0.793146
132	0.221100	0.327860	0.950994	0.422338	0.407918	0.274432	0.794243
133	0.221400	0.329471	0.950689	0.421118	0.406700	0.273221	0.795171
134	0.221400	0.327587	0.951063	0.422652	0.407992	0.274605	0.793366
135	0.221200	0.327723	0.951440	0.423881	0.409120	0.275921	0.790937

```
TrainOutput(global step=5400, training loss=0.34028113382833974,
```

```
print(trainer2.state.best_metric)
print(trainer2.state.best_model_checkpoint)
```

```
0.3261089026927948
training/visnr_aug/p2/checkpoint-4800
```

```
torch.save(trainer2.model.state_dict(), 'visnrfinal.pt')
```

```
from monai.networks.utils import one_hot
```

```
dice_mm = mm.DiceMetric(include_background=False, reduction = 'mean', get_not_nans=F
```

```
def get_metrics(y_pred, y_true):

    # for sklearn
    y_pred_np = y_pred.numpy().ravel().astype(int)
    y_true_np = y_true.numpy().ravel().astype(int)

    y_pred_tensor = y_pred
    y_true_tensor = y_true

    # compute metrics
    acc = accuracy_score(y_pred_np, y_true_np)
    f1 = f1_score(y_pred_np, y_true_np, zero_division = 0)
    prec = precision_score(y_pred_np, y_true_np, zero_division = 0)
    rec = recall_score(y_pred_np, y_true_np, zero_division = 0)
    js = jaccard_score(y_pred_np, y_true_np, zero_division = 0)

    # dice and hausdorff scores
    # pred_fg = y_pred
    # pred_bg = 1 - y_pred
    # pred_dice = torch.cat([pred_bg, pred_fg], dim = 1)

    # dice_score = DiceScore(num_classes = 2, include_background = False)
    # dice = dice_score(y_pred_tensor, y_true_tensor)

    y_pred_1hot = one_hot(y_pred, num_classes = 2)
    y_true_1hot = one_hot(y_true, num_classes = 2)

    dice_score = dice_mm(y_pred_1hot, y_true_1hot)
    dice = dice_mm.aggregate().item()






    # hausdorff = HausdorffDistance(num_classes = 2, include_background = False,
    #                               distance_metric = 'euclidean', directed = False)
    # hd = hausdorff(y_pred_tensor, y_true_tensor)

    res = {'acc': acc, 'dice': dice, 'f1': f1, 'rec': rec, 'prec': prec, 'jacc': js }

    return res
```

```
# visnr.to(device = 'cpu')
visnrp2.to(device = 'cpu')
y_pred = visnrp2.predict(test_x, threshold = 0.7732)
y = test_y
```

```
# os.chdir('/content/drive/MyDrive/XRA')
os.getcwd()
```

```
'/content/drive/MyDrive/XRA'
```

```
# os.chdir('/content/drive/MyDrive/XRA')
os.makedirs('final_pred_masks')
out_dir = 'final_pred_masks'


for i in range(len(y_pred)):

    pred_mask = y_pred[i].squeeze().numpy()

    plt.imshow(pred_mask, cmap = 'gray',
               vmax = 1.0, vmin = 0.0, interpolation = 'nearest')

    plt.axis('off')
    plt.subplots_adjust(left = 0, right = 1, bottom = 0, top = 1)

    plt.savefig(os.path.join(out_dir, f"pred_{i}.png"))
    plt.close()
```

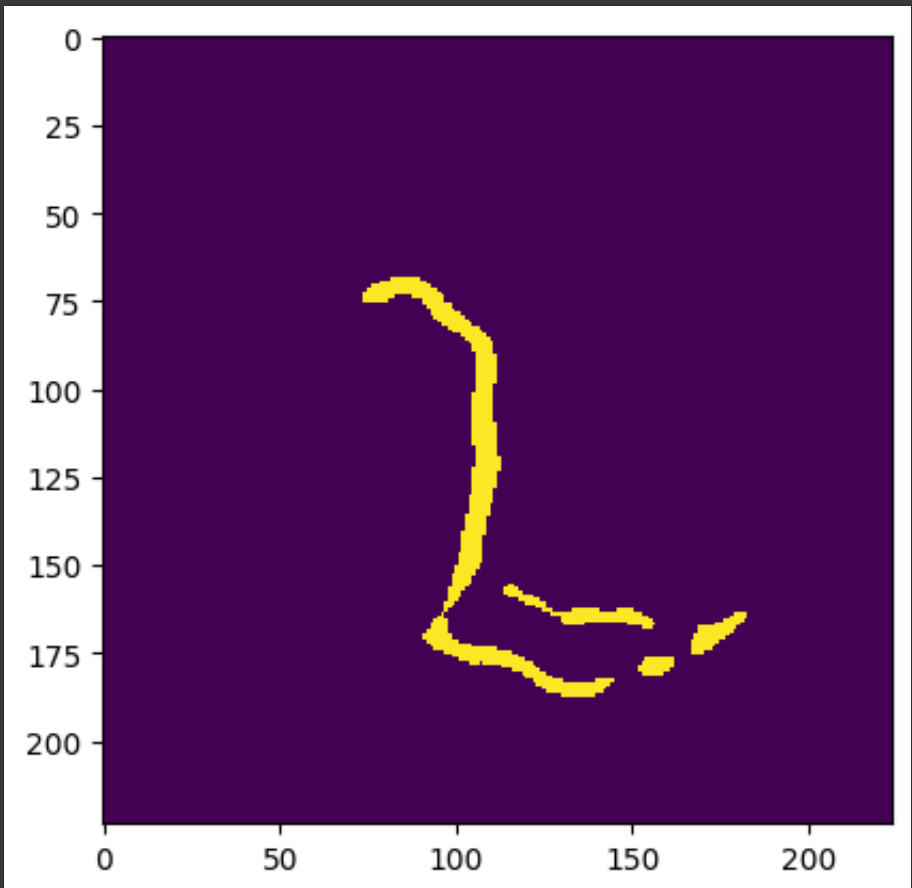
```
get_metrics(y_pred, y)
```

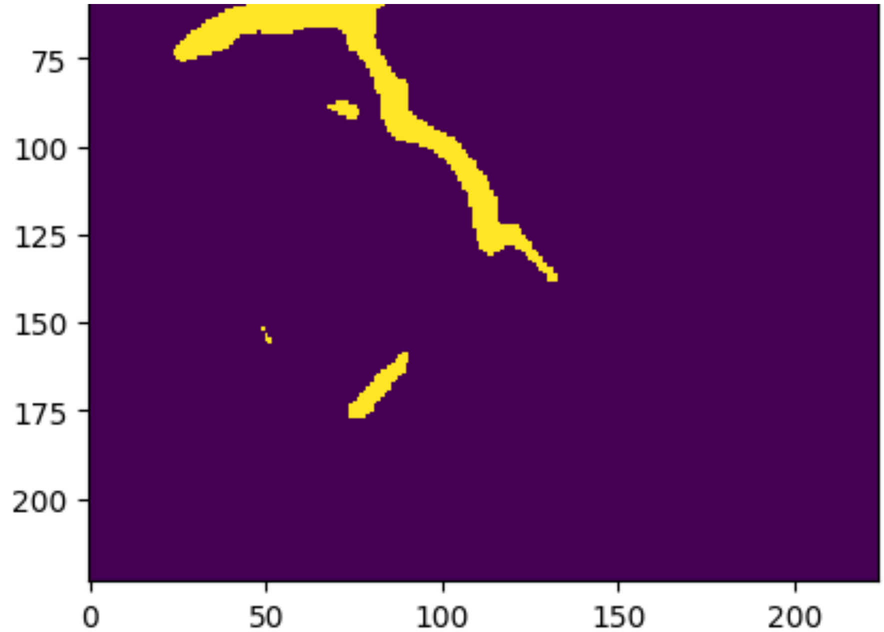
```
{'acc': 0.9733312739158163,
 'dice': 0.5225915312767029,
 'f1': 0.5336480767465878,
 'rec': 0.4235391307715145,
 'prec': 0.7211202160057769,
 'jacc': np.float64(0.36392905978707796)}
```

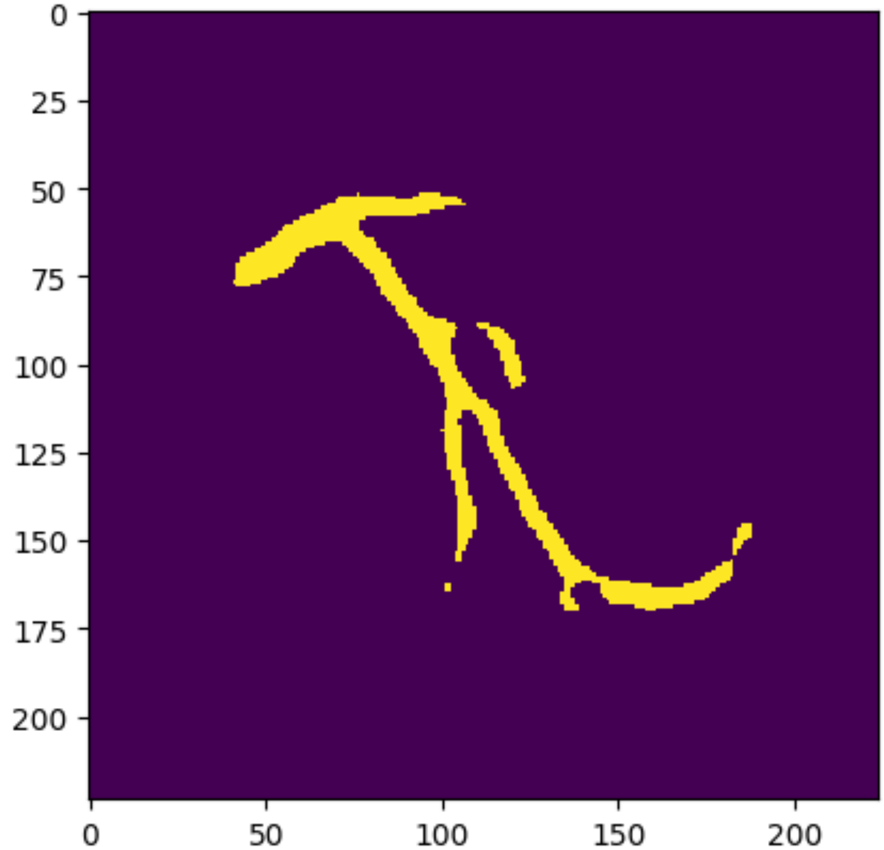
```
y_pred.shape
```

```
torch.Size([300, 1, 224, 224])
```

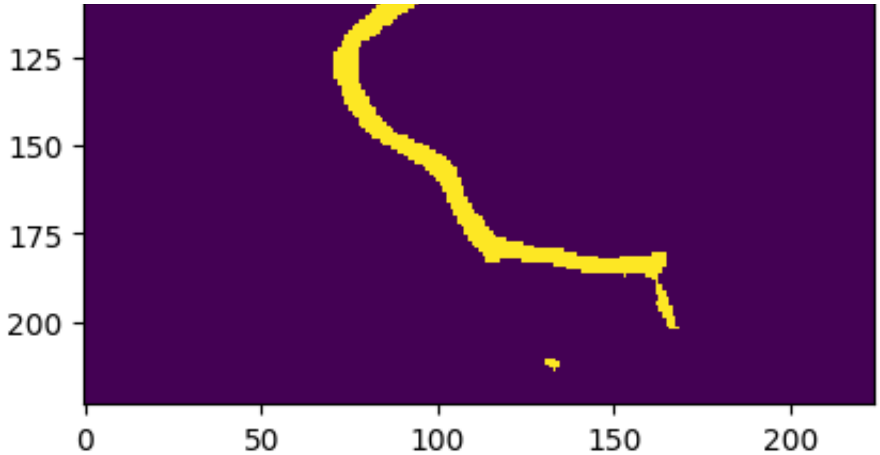
```
color_sequences = ['viridis', 'plasma', 'inferno', 'magma', 'cividis']
for i in range(10):
    pred_mask = y_pred[i].squeeze().numpy()
    plt.imshow(pred_mask, cmap = 'viridis',
               vmax = 1.0, vmin = 0.0, interpolation = 'nearest')
    plt.show()
```

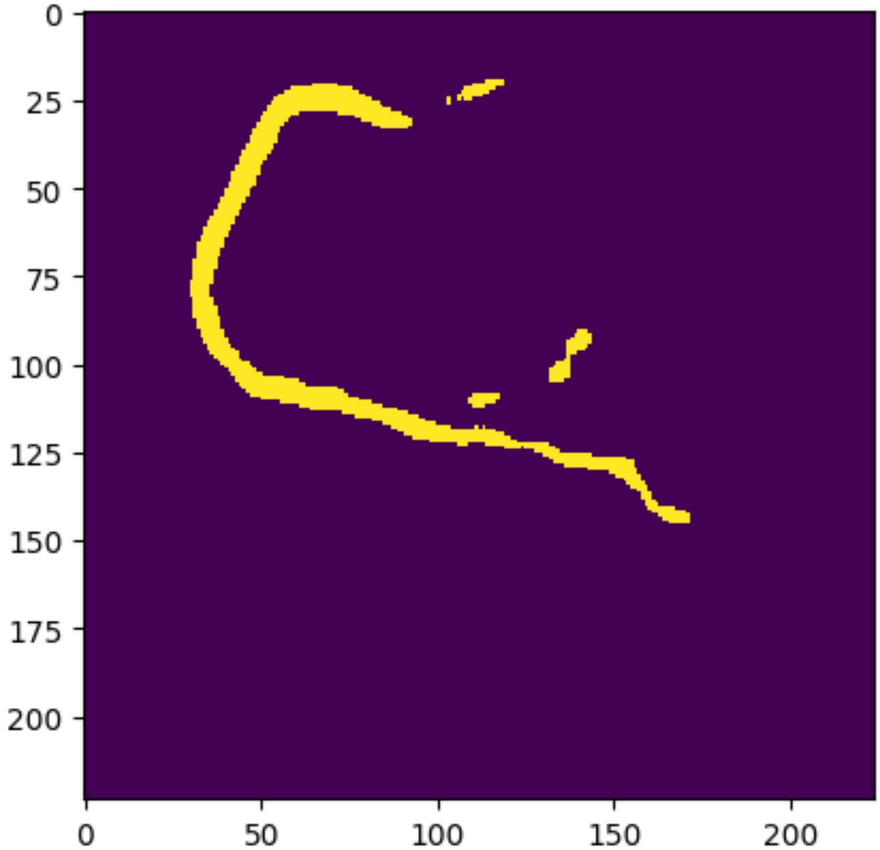



75
100
125
150
175
200
0
50
100
150
200

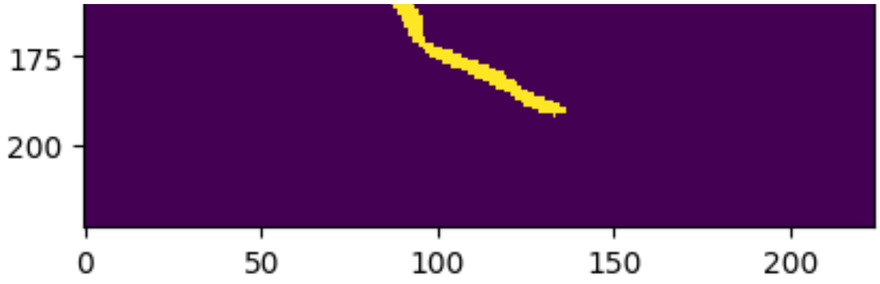

0
25
50
75
100
125
150
175
200
0
50
100
150
200

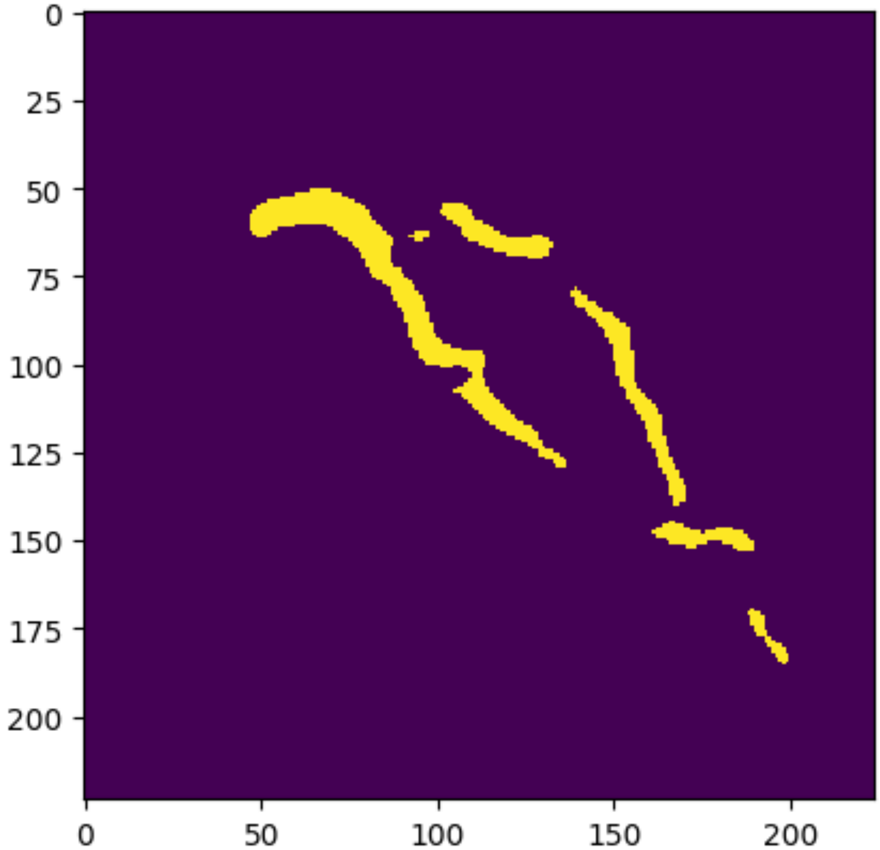

0
25
50
75
100

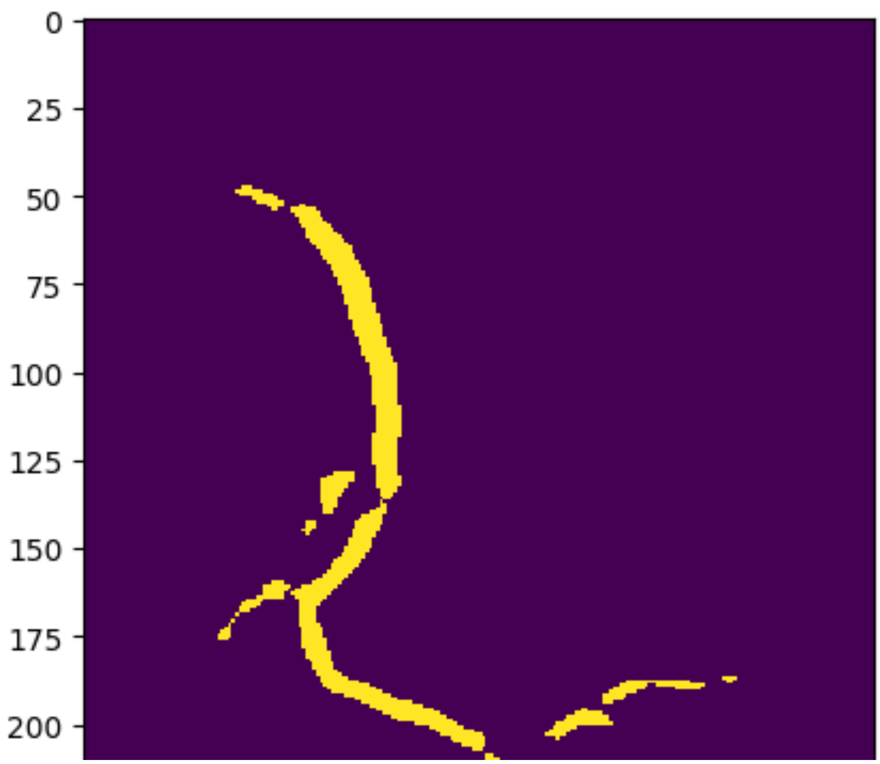

125
150
175
200
0
50
100
150
200

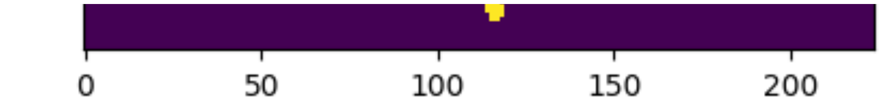
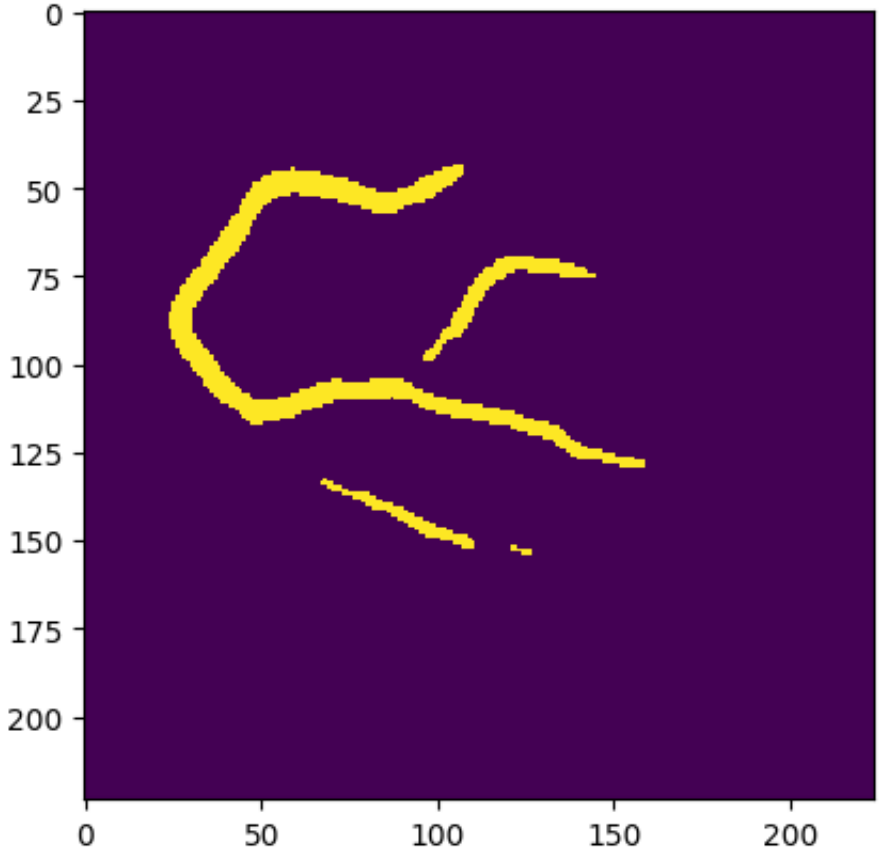
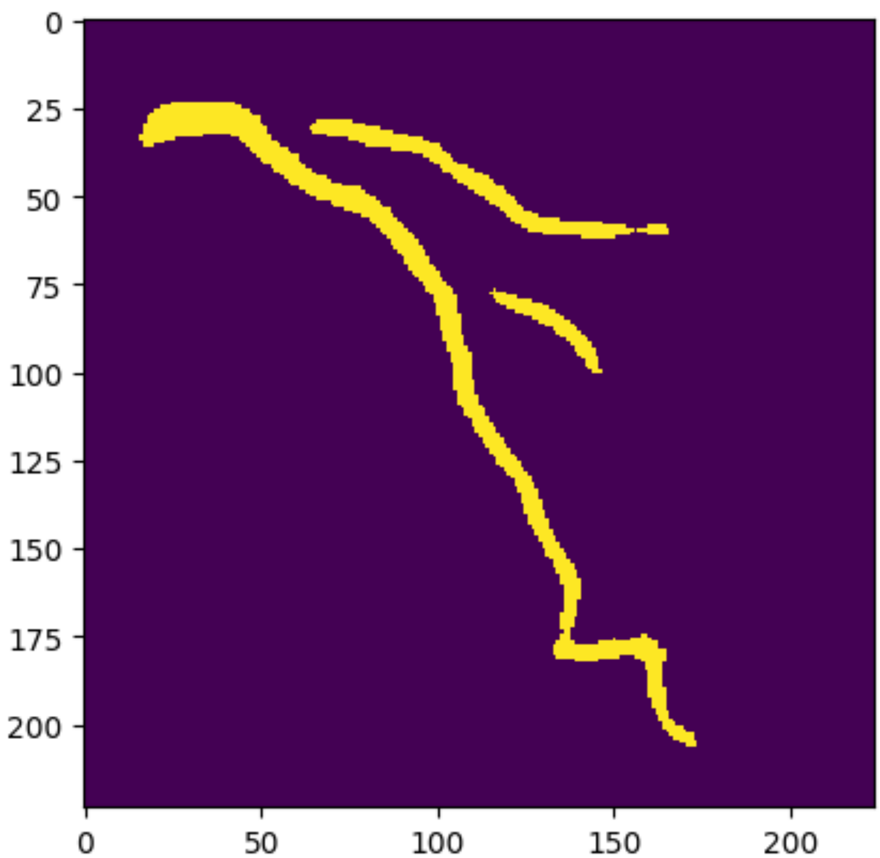

0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150


175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200


0
50
100
150
200

0
25
50
75
100
125
150
175
200
0
50
100
150
200

0
25
50
75
100
125
150
175
200
0
50
100
150
200

```
# os.chdir('/content/drive/MyDrive/XRA')
out_dir = 'vit_pred_masks'
os.makedirs('vit_pred_masks')

for i in range(len(y_pred)):

    pred_mask = y_pred[i].squeeze().numpy()

    plt.imshow(pred_mask, cmap = 'gray',
               vmax = 1.0, vmin = 0.0, interpolation = 'nearest')

    plt.axis('off')
    plt.subplots_adjust(left = 0, right = 1, bottom = 0, top = 1)

    plt.savefig(os.path.join(out_dir, f"pred_{i}.png"))
    plt.close()
```