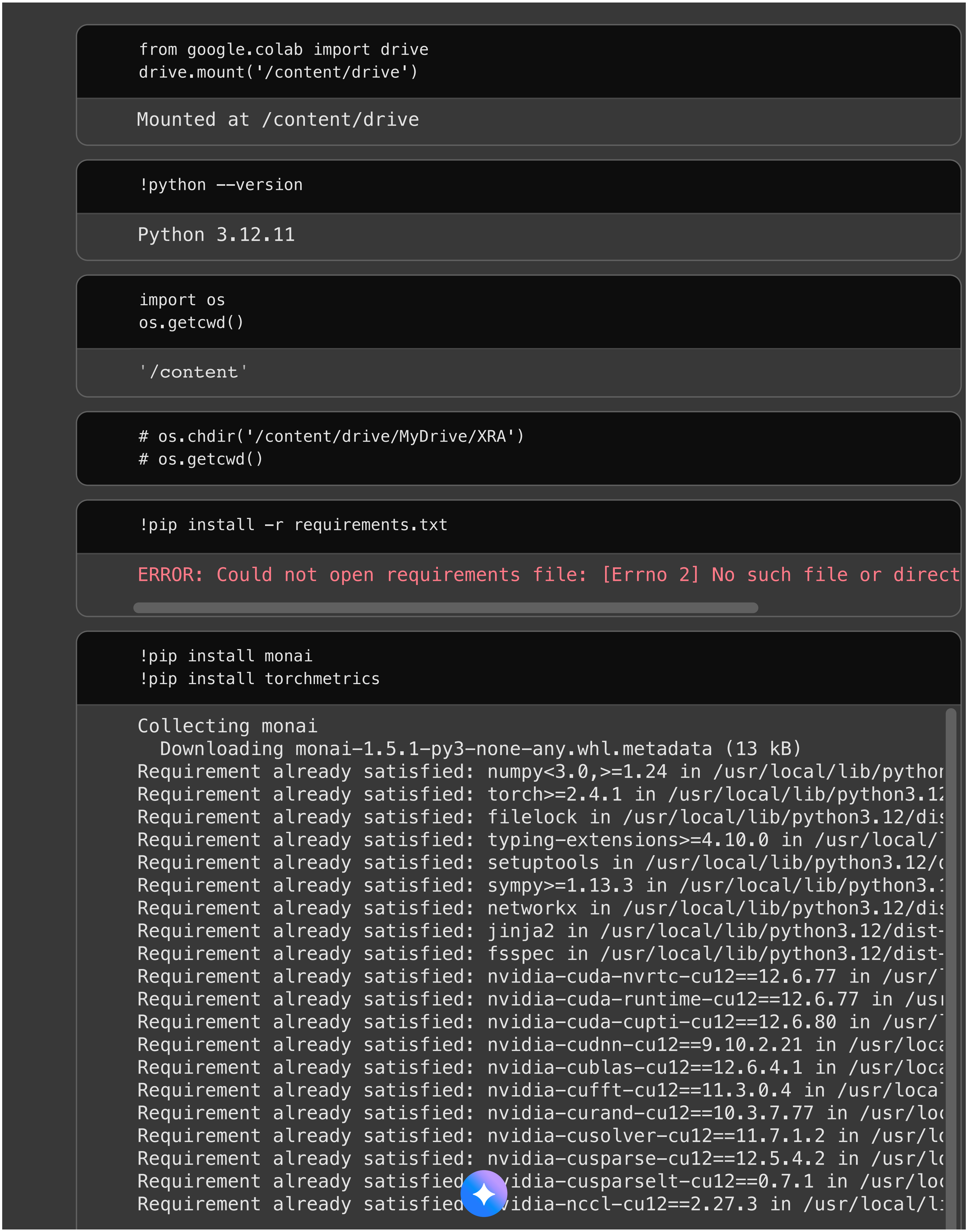


```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!python --version
```

```
Python 3.12.11
```

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

```
'/content'
```

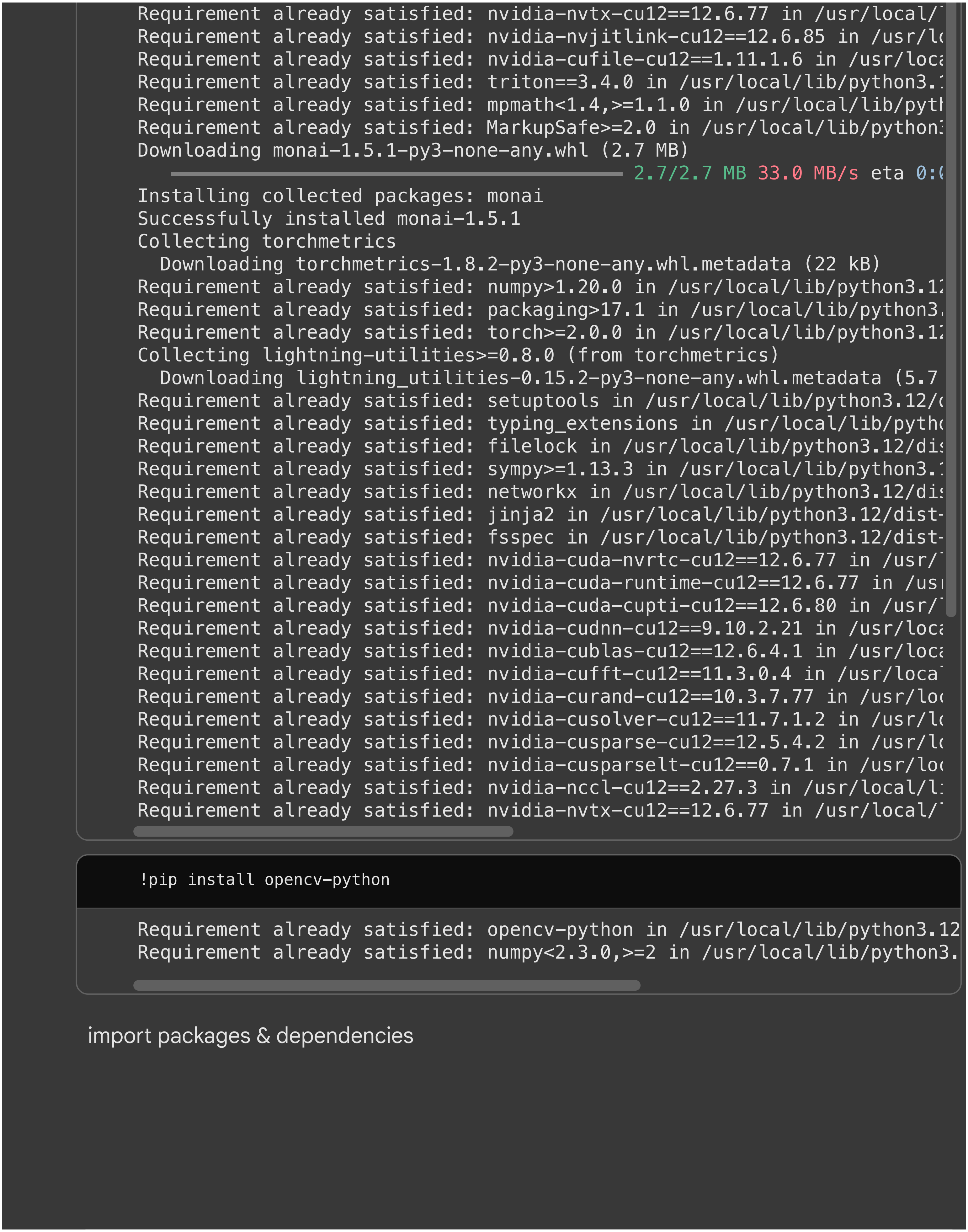
```
# os.chdir('/content/drive/MyDrive/XRA')
# os.getcwd()
```

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

```
ERROR: Could not open requirements file: [Errno 2] No such file or direct
```

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

```
Collecting monai
  Downloading monai-1.5.1-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: numpy<3.0,>=1.24 in /usr/local/lib/python
Requirement already satisfied: torch>=2.4.1 in /usr/local/lib/python3.12
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dis
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dis
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /us
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/loca
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/loca
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/loca
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/lo
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/l
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/l
Requirement already satisfied   idia-cusparselt-cu12==0.7.1 in /usr/loc
Requirement already satisfied   vidia-nccl-cu12==2.27.3 in /usr/local/li
```



```
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/l
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/loca
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python
Downloading monai-1.5.1-py3-none-any.whl (2.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 33.0 MB/s eta 0:
Installing collected packages: monai
Successfully installed monai-1.5.1
Collecting torchmetrics
  Downloading torchmetrics-1.8.2-py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.1
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.1
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.15.2-py3-none-any.whl.metadata (5.7
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/
Requirement already satisfied: typing_extensions in /usr/local/lib/pyth
Requirement already satisfied: filelock in /usr/local/lib/python3.12/di
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.
Requirement already satisfied: networkx in /usr/local/lib/python3.12/di
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /us
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/loca
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/loca
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/loca
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/lo
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/l
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/l
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/lo
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/l
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/
```

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

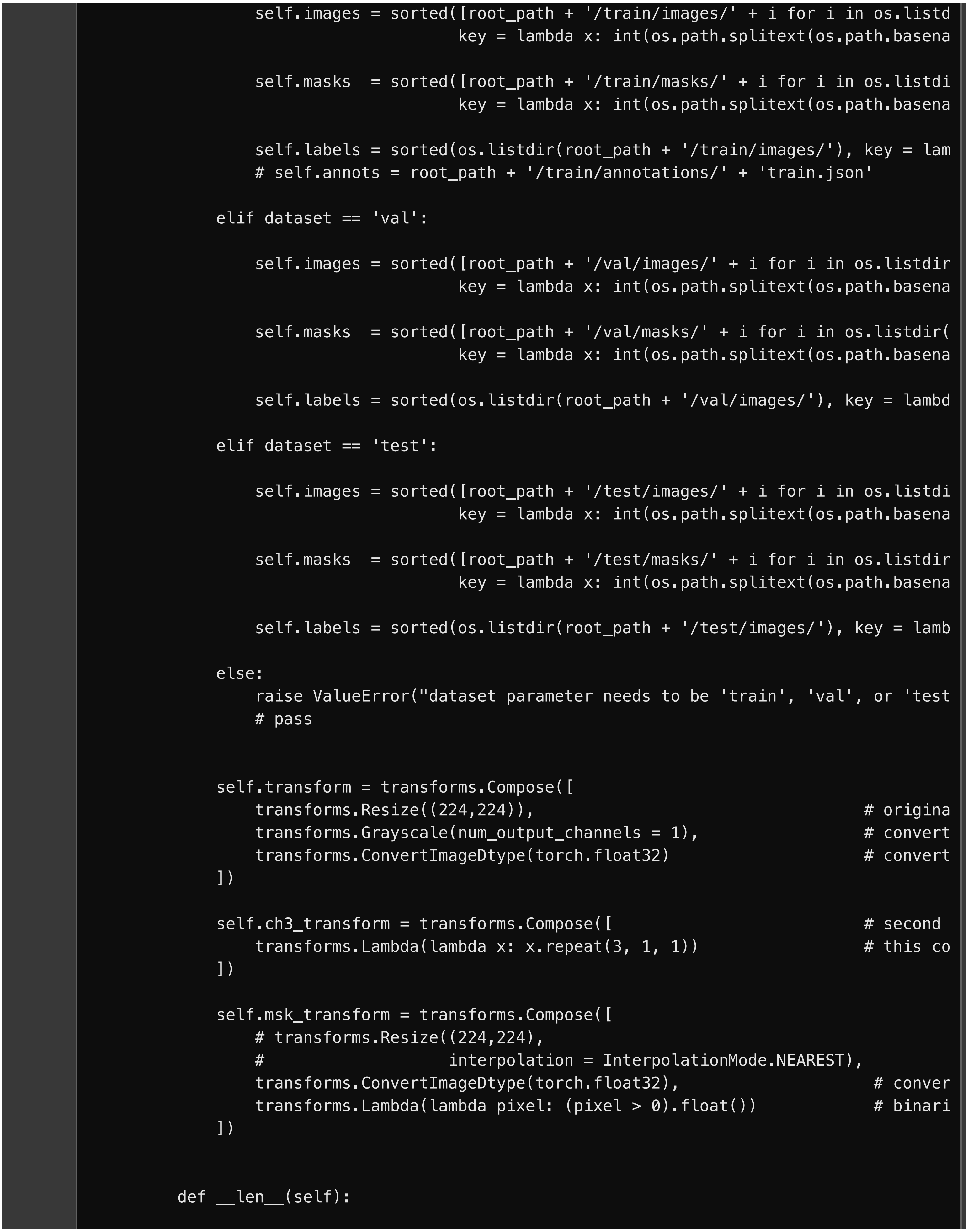
```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.12
Requirement already satisfied: numpy<2.3.0,>=2 in /usr/local/lib/python3.
```

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')
```

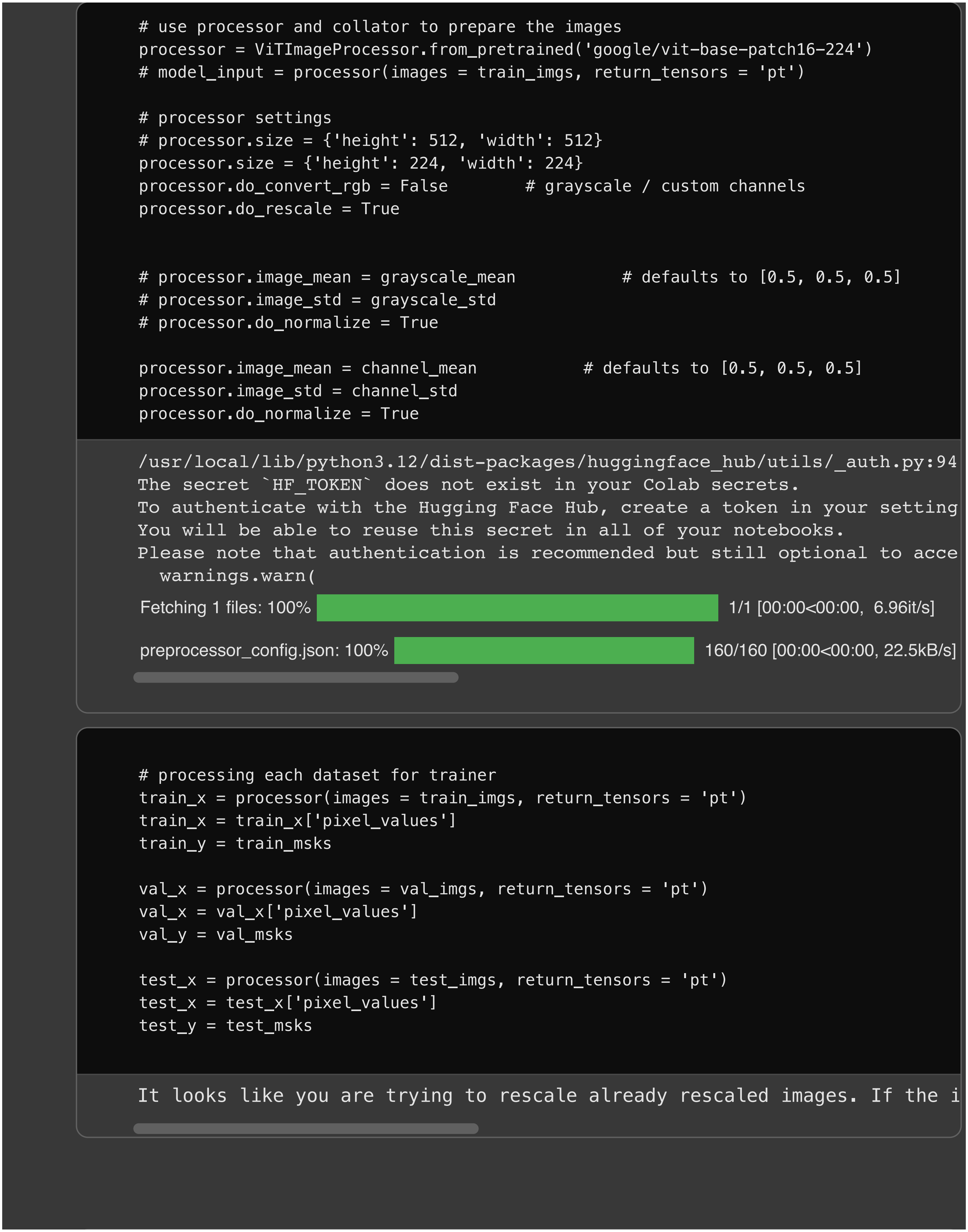
```
# define data loaders
BATCHN = [10, 20, 25, 50, 100]                       # different batch sizes batch s

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)
```

```
# 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)
```

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

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

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

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

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



```
# 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
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your setting
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to acce
  warnings.warn(
Fetching 1 files: 100%  1/1 [00:00<00:00,  6.96it/s]
preprocessor_config.json: 100%  160/160 [00:00<00:00, 22.5kB/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

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
```

```
It looks like you are trying to rescale already rescaled images. If the i
```

```
# 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)
```

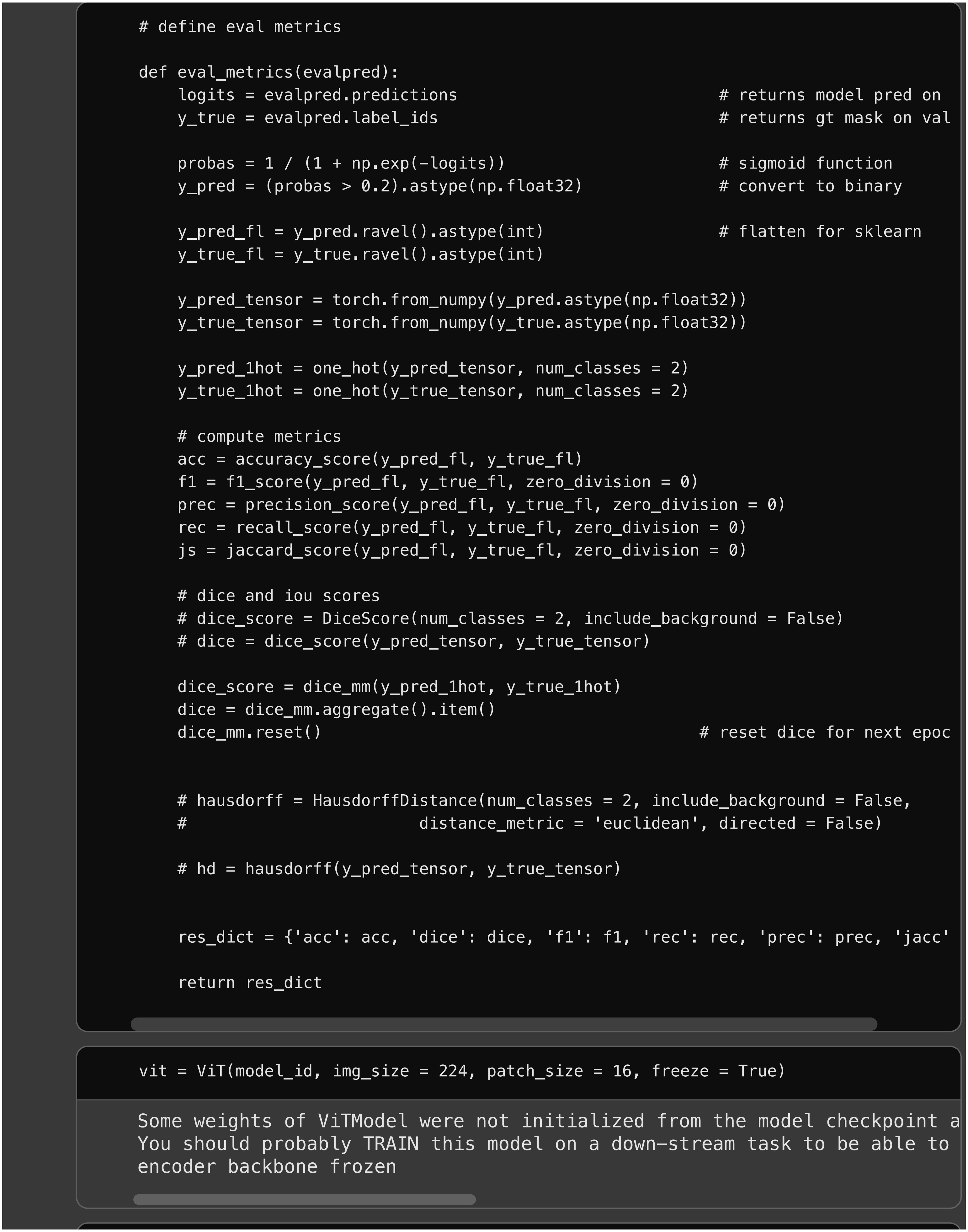
```
config.json: 🟩 69.7k/? [00:00<00:00, 7.37MB/s]

model.safetensors: 100% 🟩 346M/346M [00:01<00:00, 316MB/s]
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 ViT(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 i
        self.grid_size = (img_size // patch_size) ** 2      # grid size of each patc

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


        # define decoder
        self.decoder = nn.Sequential(
            nn.LayerNorm(768),                                # input dim [B, grid_s
            nn.Linear(768, 1024),                             # fc layer to [B, 196,
            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, pad
            # 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, r
        # self.loss_dice = ml.dice.DiceLoss(sigmoid = True, reduction = 'mean', smoo

        # self.a = 0.4

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

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



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


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

        batch_size = patch_features.shape[0]                                # return

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


        ups_logits = self.upsample(spatial_logits)                          # upsamp


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

        else:
            return ups_logits                                               # return


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

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

Training

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



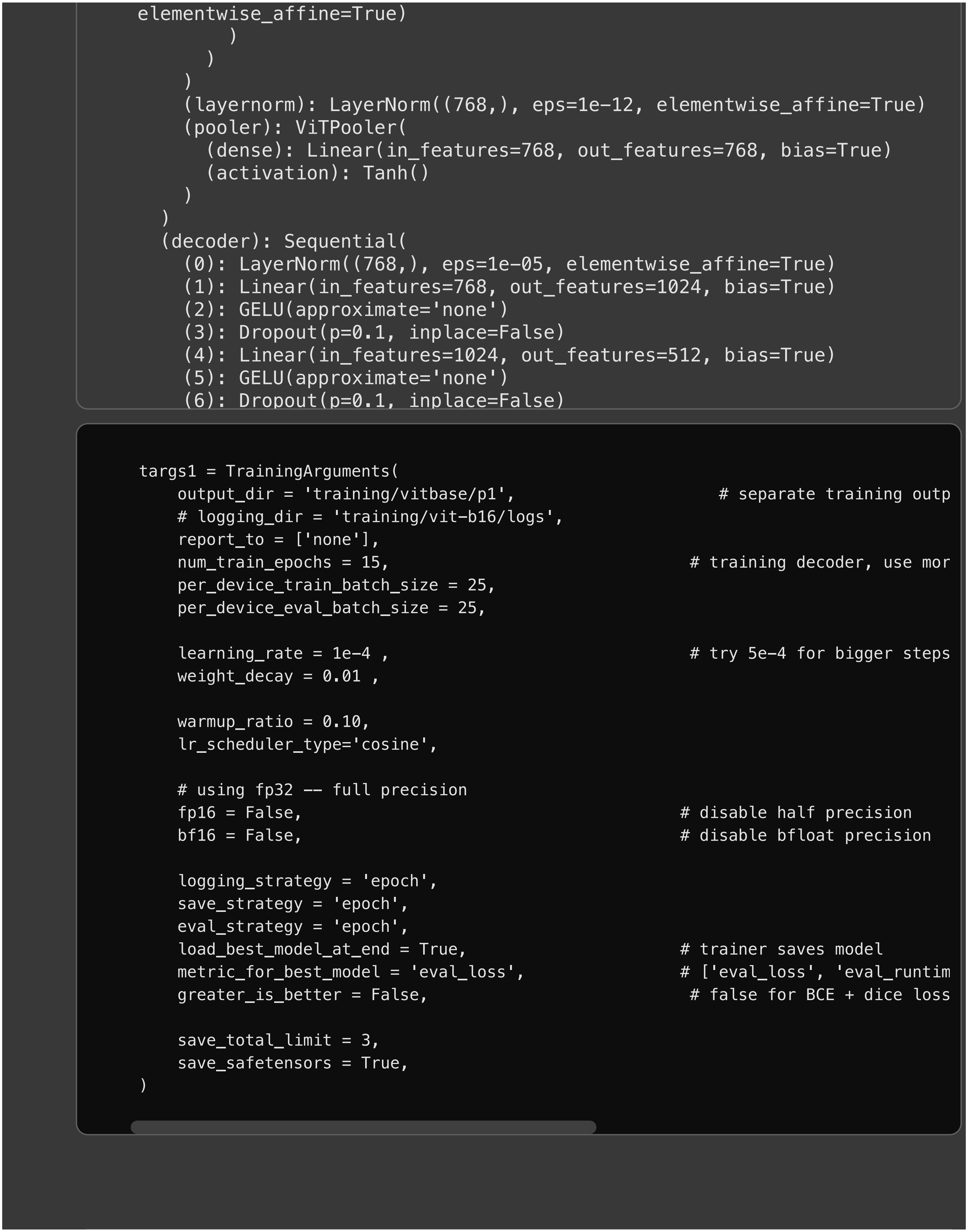
```
# define eval metrics

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
```

```
vit = ViT(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)

vit.to(device)
```

```
cpu
ViT(
  (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): ViTSelfAttention(
              (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,
```

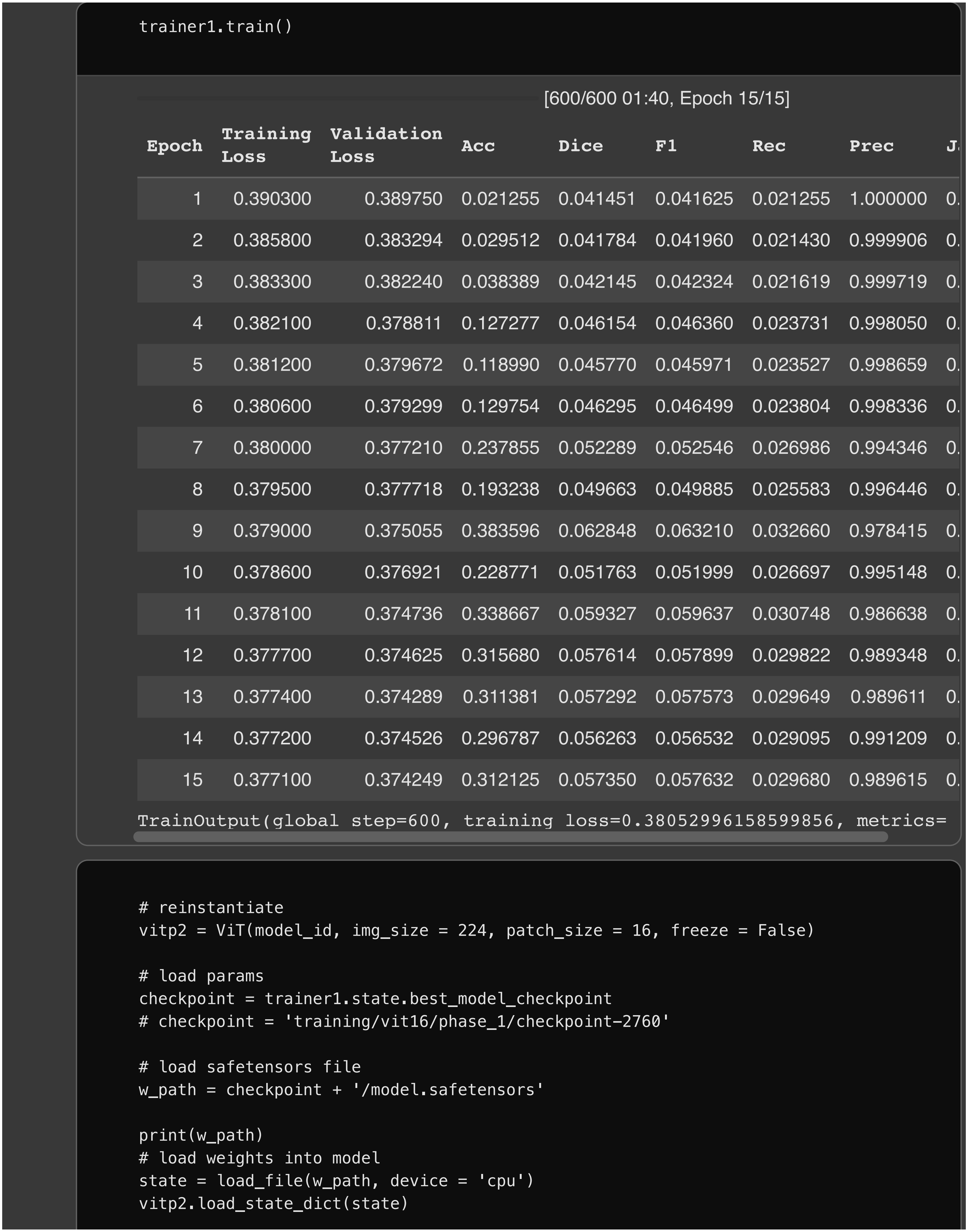


```
elementwise_affine=True)
        )
      )
    )
    (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (pooler): ViTPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (decoder): Sequential(
    (0): LayerNorm((768,), 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/vitbase/p1',                     # separate training outp
    # 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 = vit,
    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 01:40, Epoch 15/15]

Epoch	Training Loss	Validation Loss	Acc	Dice	F1	Rec	Prec	J
1	0.390300	0.389750	0.021255	0.041451	0.041625	0.021255	1.000000	0.
2	0.385800	0.383294	0.029512	0.041784	0.041960	0.021430	0.999906	0.
3	0.383300	0.382240	0.038389	0.042145	0.042324	0.021619	0.999719	0.
4	0.382100	0.378811	0.127277	0.046154	0.046360	0.023731	0.998050	0.
5	0.381200	0.379672	0.118990	0.045770	0.045971	0.023527	0.998659	0.
6	0.380600	0.379299	0.129754	0.046295	0.046499	0.023804	0.998336	0.
7	0.380000	0.377210	0.237855	0.052289	0.052546	0.026986	0.994346	0.
8	0.379500	0.377718	0.193238	0.049663	0.049885	0.025583	0.996446	0.
9	0.379000	0.375055	0.383596	0.062848	0.063210	0.032660	0.978415	0.
10	0.378600	0.376921	0.228771	0.051763	0.051999	0.026697	0.995148	0.
11	0.378100	0.374736	0.338667	0.059327	0.059637	0.030748	0.986638	0.
12	0.377700	0.374625	0.315680	0.057614	0.057899	0.029822	0.989348	0.
13	0.377400	0.374289	0.311381	0.057292	0.057573	0.029649	0.989611	0.
14	0.377200	0.374526	0.296787	0.056263	0.056532	0.029095	0.991209	0.
15	0.377100	0.374249	0.312125	0.057350	0.057632	0.029680	0.989615	0.

```
TrainOutput(global_step=600, training_loss=0.38052996158599856, metrics=
```

```
# reinstantiate
vitp2 = ViT(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')
vitp2.load_state_dict(state)
```

```
print(device)
vitp2.to(device)
```

```
Some weights of ViTModel were not initialized from the model checkpoint
You should probably TRAIN this model on a down-stream task to be able to
encoder backbone training enabled
training/vitbase/p1/checkpoint-600/model.safetensors
cpu
ViT(
  (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): ViTSelfAttention(
              (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)
```

```
targs2 = TrainingArguments(
    output_dir = 'training/vitp2/p2',                     # separate training output
    # logging_dir = 'training/vit-b16/logs',
    report_to = ['none'],
    num_train_epochs = 100,                               # 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,
)
```

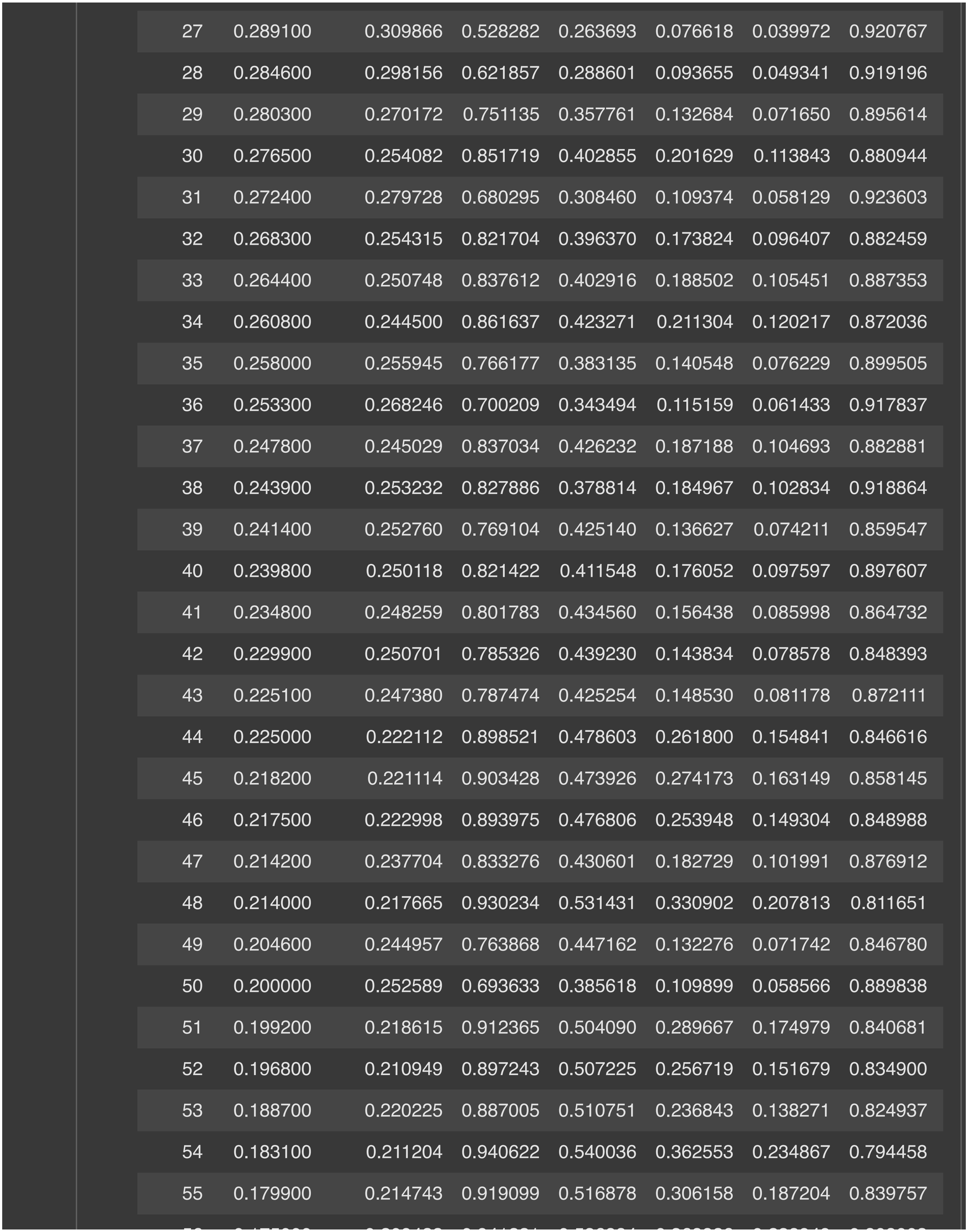
```
trainer2 = Trainer(
    model = vitp2,
    args = targs2,
    train_dataset = ds_train,
    eval_dataset = ds_val,
    compute_metrics = eval_metrics,
    callbacks = [EarlyStoppingCallback(early_stopping_patience = 30)]
)
```

```
trainer2.train()
```

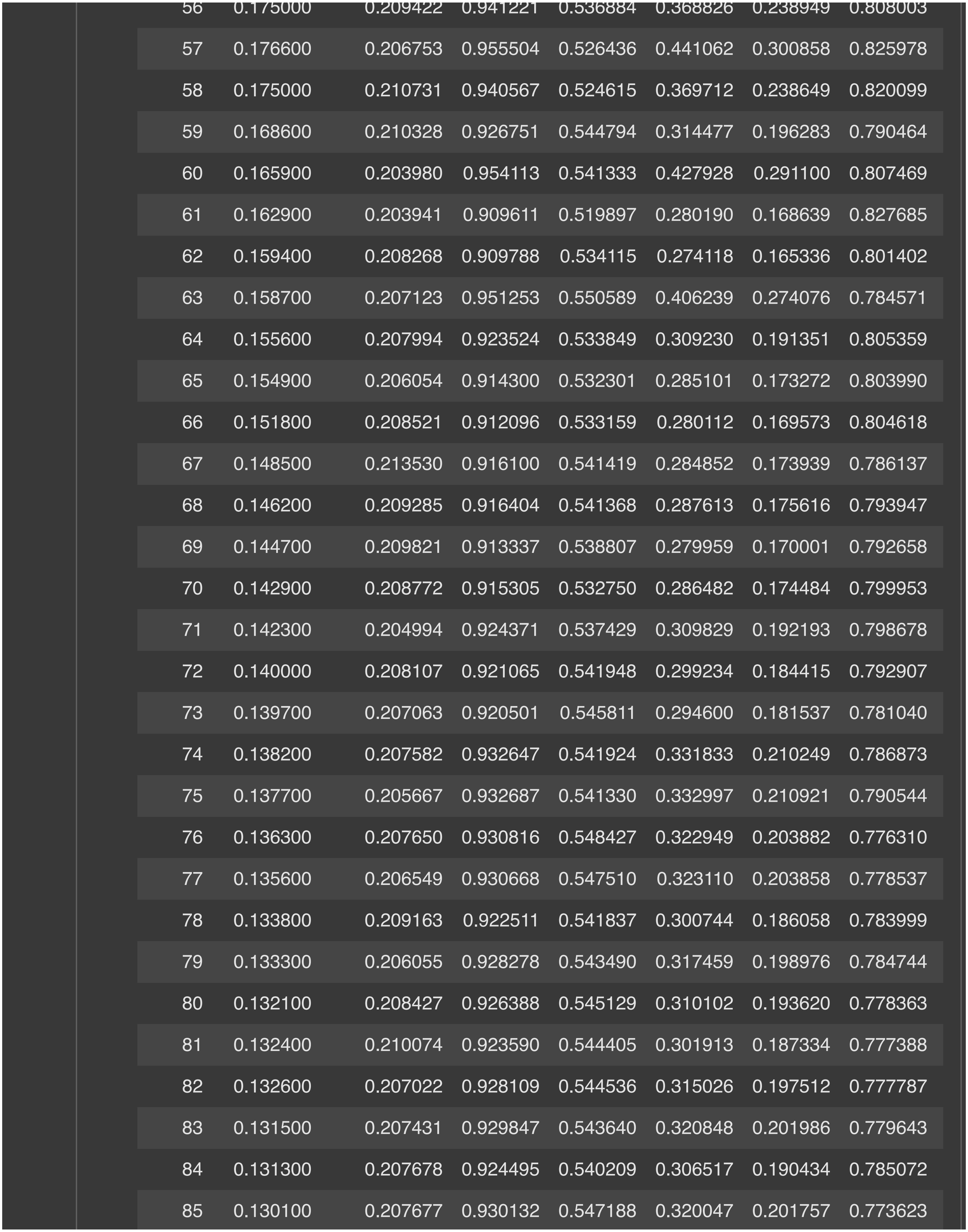
```
trainer2.train()
```

[3640/4000 18:39 < 01:50, 3.25 it/s, Epoch 91/100]

Epoch	Training Loss	Validation Loss	Acc	Dice	F1	Rec	Prec
1	0.377500	0.374256	0.353429	0.060493	0.060801	0.031369	0.984655
2	0.376600	0.387739	0.099282	0.044814	0.045006	0.023022	0.998561
3	0.375200	0.372299	0.291278	0.055929	0.056139	0.028887	0.991617
4	0.372600	0.374585	0.085245	0.044195	0.044368	0.022688	0.999076
5	0.370000	0.369869	0.159885	0.047791	0.047928	0.024556	0.994890
6	0.366200	0.363797	0.065657	0.043364	0.043494	0.022231	0.999451
7	0.362300	0.353678	0.358062	0.061564	0.061686	0.031832	0.992766
8	0.358800	0.361820	0.072629	0.043931	0.043818	0.022400	0.999733
9	0.355100	0.350454	0.278476	0.058576	0.055436	0.028511	0.996160
10	0.351700	0.346976	0.407262	0.085932	0.066124	0.034208	0.987304
11	0.348400	0.322476	0.879974	0.268295	0.239593	0.138438	0.889646
12	0.346100	0.357123	0.230282	0.084222	0.051736	0.026564	0.987895
13	0.341900	0.354670	0.283212	0.087001	0.055376	0.028486	0.988490
14	0.338200	0.335760	0.479340	0.157711	0.073187	0.038032	0.967186
15	0.333400	0.310782	0.647568	0.229161	0.101956	0.053897	0.941255
16	0.330100	0.324500	0.517167	0.230967	0.075108	0.039148	0.922375
17	0.326200	0.333002	0.432570	0.197307	0.066465	0.034437	0.950360
18	0.322600	0.317715	0.640068	0.277604	0.095313	0.050346	0.892046
19	0.317900	0.335726	0.393300	0.183173	0.062570	0.032347	0.952615
20	0.313100	0.334751	0.404473	0.191256	0.063438	0.032816	0.948916
21	0.310100	0.318140	0.475516	0.226961	0.071033	0.036906	0.943421
22	0.306500	0.356484	0.184189	0.112424	0.048536	0.024885	0.978992
23	0.303100	0.303341	0.586903	0.269088	0.086856	0.045569	0.924330
24	0.298400	0.317133	0.474759	0.228177	0.070929	0.036850	0.943304
25	0.295100	0.297112	0.955981	0.471389	0.387503	0.275116	0.655130
26	0.295000	0.276304	0.749190	0.328234	0.133967	0.072289	0.912694



27	0.289100	0.309866	0.528282	0.263693	0.076618	0.039972	0.920767
28	0.284600	0.298156	0.621857	0.288601	0.093655	0.049341	0.919196
29	0.280300	0.270172	0.751135	0.357761	0.132684	0.071650	0.895614
30	0.276500	0.254082	0.851719	0.402855	0.201629	0.113843	0.880944
31	0.272400	0.279728	0.680295	0.308460	0.109374	0.058129	0.923603
32	0.268300	0.254315	0.821704	0.396370	0.173824	0.096407	0.882459
33	0.264400	0.250748	0.837612	0.402916	0.188502	0.105451	0.887353
34	0.260800	0.244500	0.861637	0.423271	0.211304	0.120217	0.872036
35	0.258000	0.255945	0.766177	0.383135	0.140548	0.076229	0.899505
36	0.253300	0.268246	0.700209	0.343494	0.115159	0.061433	0.917837
37	0.247800	0.245029	0.837034	0.426232	0.187188	0.104693	0.882881
38	0.243900	0.253232	0.827886	0.378814	0.184967	0.102834	0.918864
39	0.241400	0.252760	0.769104	0.425140	0.136627	0.074211	0.859547
40	0.239800	0.250118	0.821422	0.411548	0.176052	0.097597	0.897607
41	0.234800	0.248259	0.801783	0.434560	0.156438	0.085998	0.864732
42	0.229900	0.250701	0.785326	0.439230	0.143834	0.078578	0.848393
43	0.225100	0.247380	0.787474	0.425254	0.148530	0.081178	0.872111
44	0.225000	0.222112	0.898521	0.478603	0.261800	0.154841	0.846616
45	0.218200	0.221114	0.903428	0.473926	0.274173	0.163149	0.858145
46	0.217500	0.222998	0.893975	0.476806	0.253948	0.149304	0.848988
47	0.214200	0.237704	0.833276	0.430601	0.182729	0.101991	0.876912
48	0.214000	0.217665	0.930234	0.531431	0.330902	0.207813	0.811651
49	0.204600	0.244957	0.763868	0.447162	0.132276	0.071742	0.846780
50	0.200000	0.252589	0.693633	0.385618	0.109899	0.058566	0.889838
51	0.199200	0.218615	0.912365	0.504090	0.289667	0.174979	0.840681
52	0.196800	0.210949	0.897243	0.507225	0.256719	0.151679	0.834900
53	0.188700	0.220225	0.887005	0.510751	0.236843	0.138271	0.824937
54	0.183100	0.211204	0.940622	0.540036	0.362553	0.234867	0.794458
55	0.179900	0.214743	0.919099	0.516878	0.306158	0.187204	0.839757
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]



56	0.175000	0.209422	0.941221	0.536884	0.368826	0.238949	0.808003
57	0.176600	0.206753	0.955504	0.526436	0.441062	0.300858	0.825978
58	0.175000	0.210731	0.940567	0.524615	0.369712	0.238649	0.820099
59	0.168600	0.210328	0.926751	0.544794	0.314477	0.196283	0.790464
60	0.165900	0.203980	0.954113	0.541333	0.427928	0.291100	0.807469
61	0.162900	0.203941	0.909611	0.519897	0.280190	0.168639	0.827685
62	0.159400	0.208268	0.909788	0.534115	0.274118	0.165336	0.801402
63	0.158700	0.207123	0.951253	0.550589	0.406239	0.274076	0.784571
64	0.155600	0.207994	0.923524	0.533849	0.309230	0.191351	0.805359
65	0.154900	0.206054	0.914300	0.532301	0.285101	0.173272	0.803990
66	0.151800	0.208521	0.912096	0.533159	0.280112	0.169573	0.804618
67	0.148500	0.213530	0.916100	0.541419	0.284852	0.173939	0.786137
68	0.146200	0.209285	0.916404	0.541368	0.287613	0.175616	0.793947
69	0.144700	0.209821	0.913337	0.538807	0.279959	0.170001	0.792658
70	0.142900	0.208772	0.915305	0.532750	0.286482	0.174484	0.799953
71	0.142300	0.204994	0.924371	0.537429	0.309829	0.192193	0.798678
72	0.140000	0.208107	0.921065	0.541948	0.299234	0.184415	0.792907
73	0.139700	0.207063	0.920501	0.545811	0.294600	0.181537	0.781040
74	0.138200	0.207582	0.932647	0.541924	0.331833	0.210249	0.786873
75	0.137700	0.205667	0.932687	0.541330	0.332997	0.210921	0.790544
76	0.136300	0.207650	0.930816	0.548427	0.322949	0.203882	0.776310
77	0.135600	0.206549	0.930668	0.547510	0.323110	0.203858	0.778537
78	0.133800	0.209163	0.922511	0.541837	0.300744	0.186058	0.783999
79	0.133300	0.206055	0.928278	0.543490	0.317459	0.198976	0.784744
80	0.132100	0.208427	0.926388	0.545129	0.310102	0.193620	0.778363
81	0.132400	0.210074	0.923590	0.544405	0.301913	0.187334	0.777388
82	0.132600	0.207022	0.928109	0.544536	0.315026	0.197512	0.777787
83	0.131500	0.207431	0.929847	0.543640	0.320848	0.201986	0.779643
84	0.131300	0.207678	0.924495	0.540209	0.306517	0.190434	0.785072
85	0.130100	0.207677	0.930132	0.547188	0.320047	0.201757	0.773623

86	0.130600	0.209666	0.923295	0.540501	0.302627	0.187557	0.783033
87	0.129500	0.207567	0.928399	0.545925	0.315408	0.197927	0.776029
88	0.129900	0.209017	0.928687	0.543926	0.316656	0.198822	0.777374
89	0.129300	0.206320	0.932492	0.543434	0.328765	0.208431	0.777824
90	0.129100	0.207300	0.930716	0.544373	0.322640	0.203635	0.776329
91	0.129300	0.210549	0.923350	0.540733	0.301790	0.187124	0.779376

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

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

```
torch.save(trainer2.model.state_dict(), 'vit_base.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')
vitp2.to(device = 'cpu')
y_pred = vitp2.predict(test_x, threshold = 0.6)
y = test_y
```

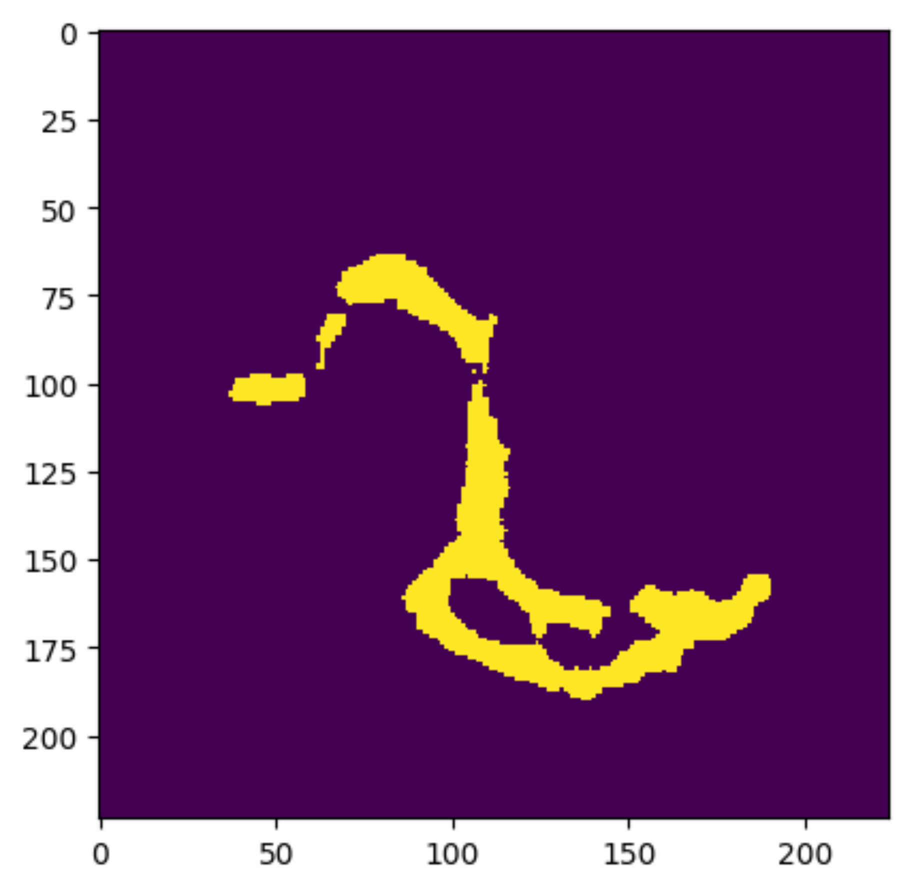
```
get_metrics(y_pred, y)
```

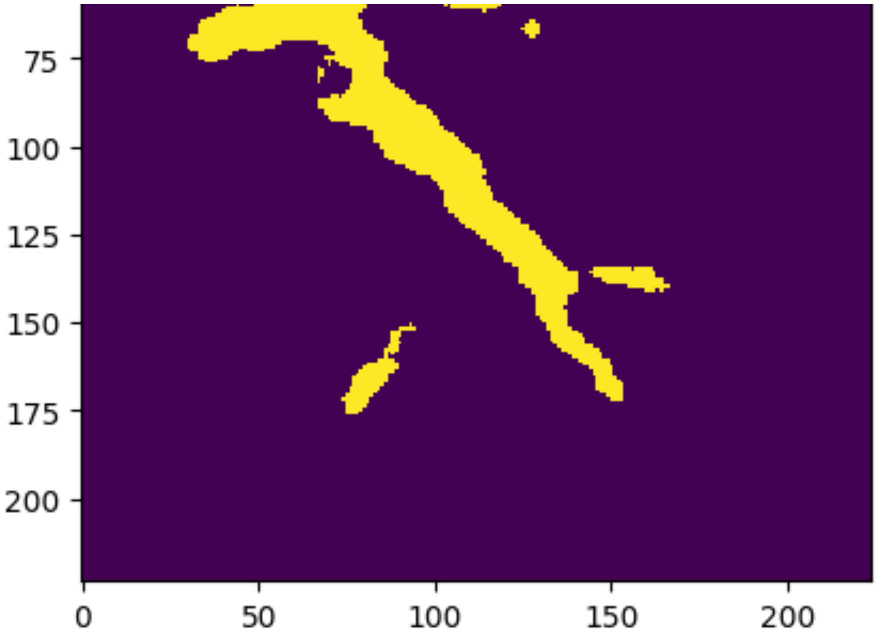
```
{'acc': 0.9617795360331632,
 'dice': 0.4230040907859802,
 'f1': 0.4290161899081285,
 'rec': 0.31365722768915083,
 'prec': 0.6785940786788484,
 'jacc': np.float64(0.2730875946347528)}
```

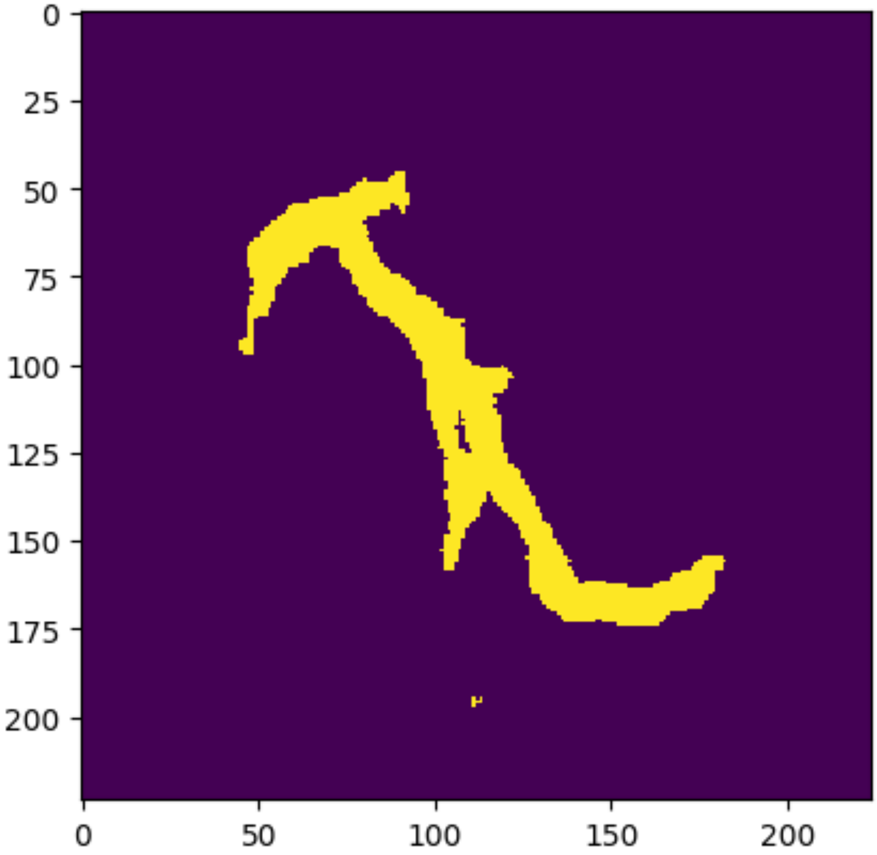
```
y_pred.shape
```

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

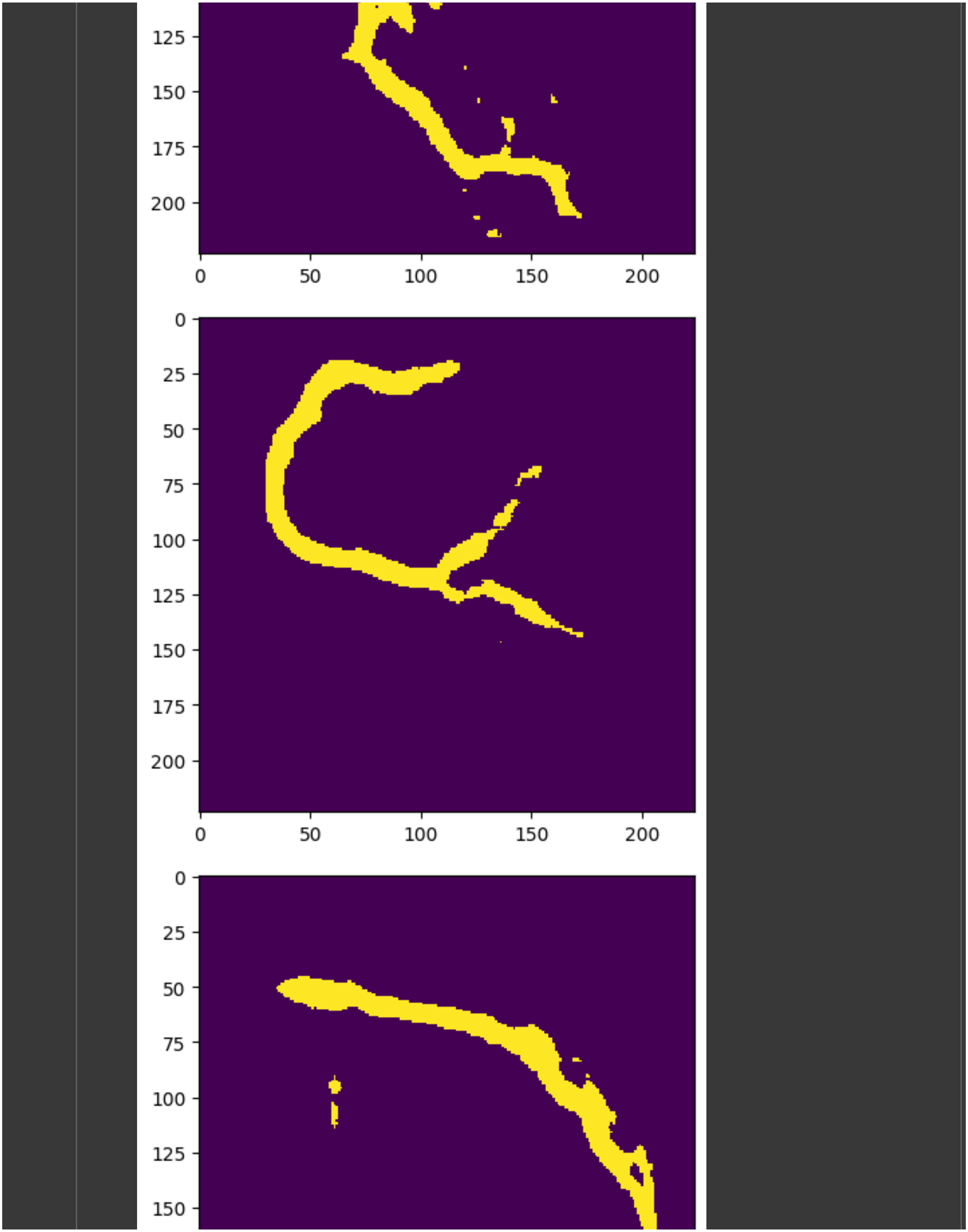
```
color_sequences = ['viridis', 'plasma', 'inferno', 'magma', 'cividis']
for i in range(10):
    pred_mask = y_pred[i].squeeze().numpy()
    plt.imshow(pred_mask, cmap = 'viridis',
               vmax = 0.75, vmin = 0.25, interpolation = 'nearest')
    plt.show()
```

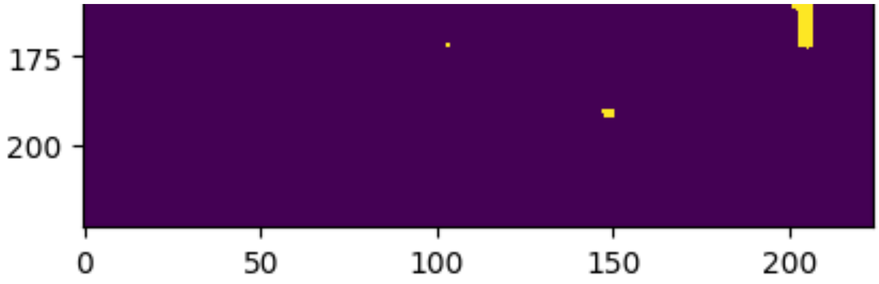


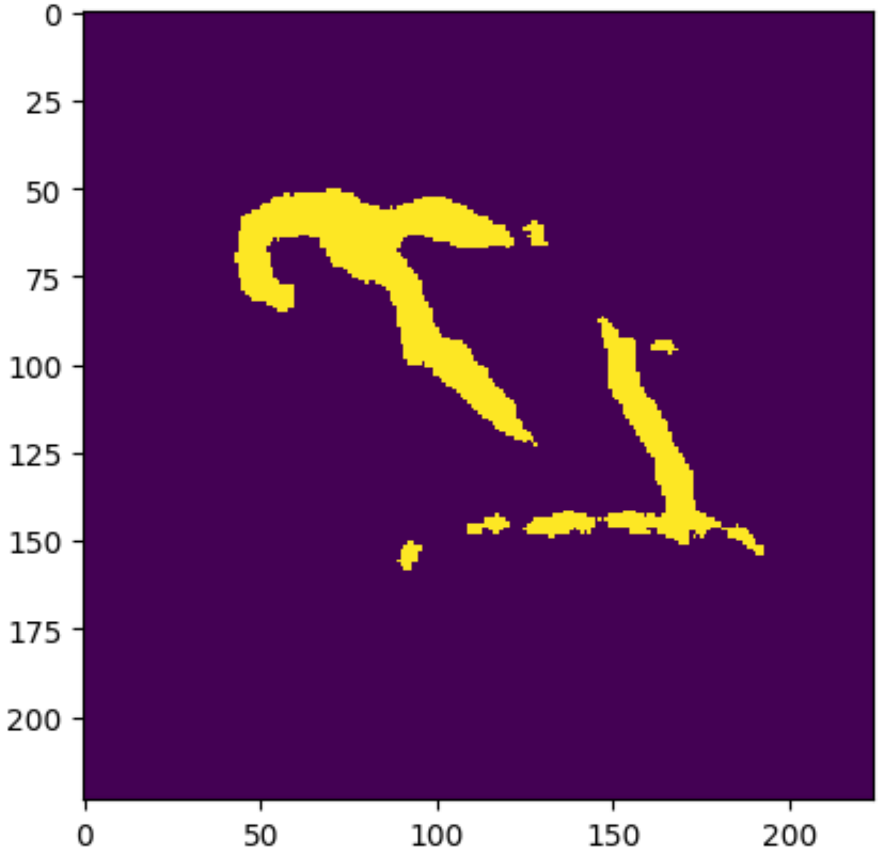

75
100
125
150
175
200
0
50
100
150
200


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

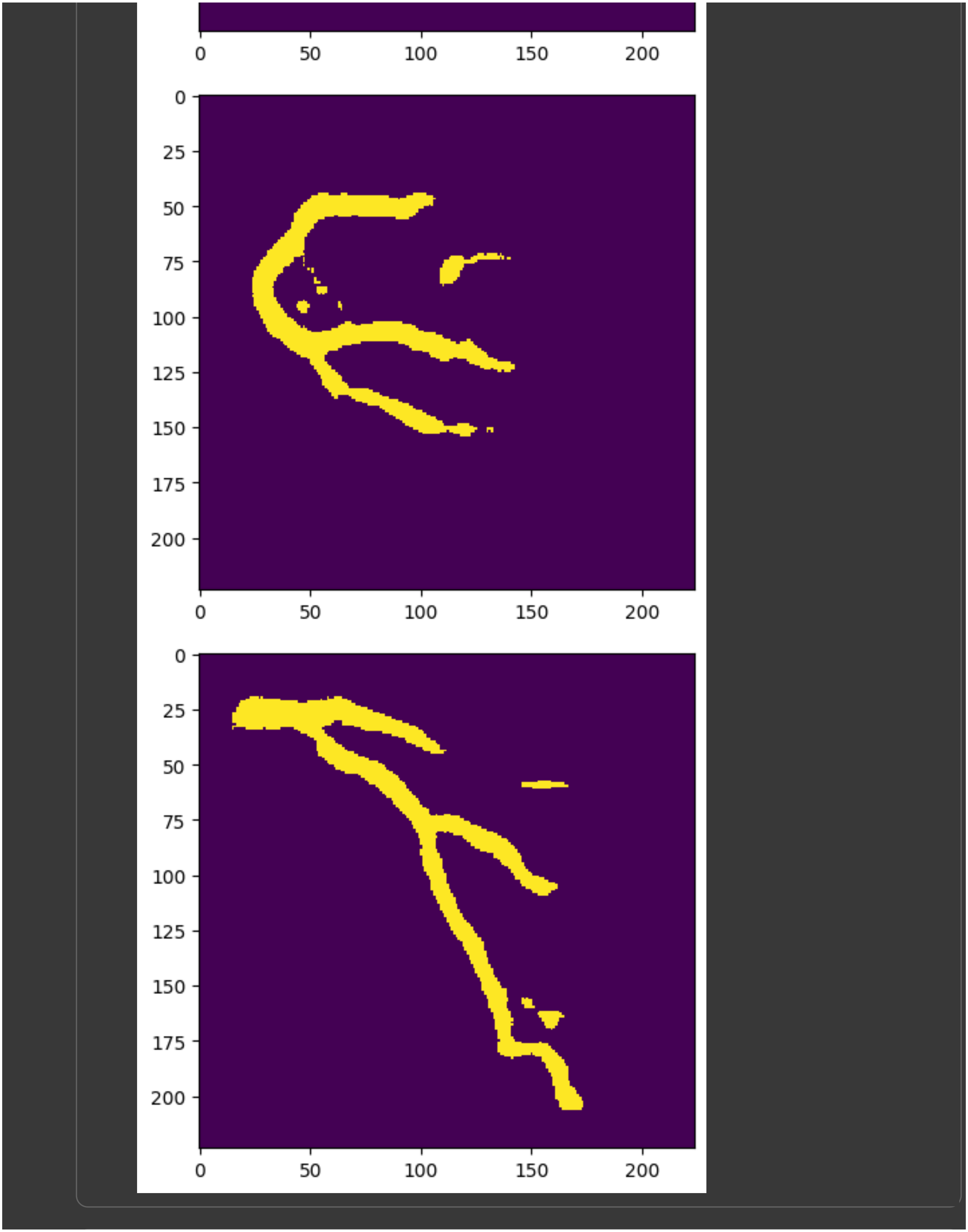
0
25
50
75
100


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')
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()
```