```
!pip install torch
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.me
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.me
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadat
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metada
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadat
Collecting nvidia-curand-cu12==10.3.5.147 (from torch)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.meta
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.meta
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.me
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.met
Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-p
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 M
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 2.9 MB/s eta 0:00:00
Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 78.3 MB/s eta 0:00:00
Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 61.3 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (8
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 53.7 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.3 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 6.7 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 17.7 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.2 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.1 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 95.1 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-c
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
```

```
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
```

```
!pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0
```

```
# Creating U-Net architecture

# page 4 of the paper: (https://arxiv.org/pdf/1505.04597.pdf)

# Network Architecture
# The network architecture is illustrated in Figure 1. It consists of a contracting
# path (left side) and an expansive path (right side). The contracting path follows
# the typical architecture of a convolutional network. It consists of the repeated
# application of two 3x3 convolutions (unpadded convolutions), each followed by
# a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2
# for downsampling. At each downsampling step we double the number of feature
# channels. Every step in the expansive path consists of an upsampling of the
# feature map followed by a 2x2 convolution ("up-convolution") that halves the
# number of feature channels, a concatenation with the correspondingly cropped
# feature map from the contracting path, and two 3x3 convolutions, each followed by a Re
# every convolution. At the final layer a 1x1 convolution is used to map each 64-
# component feature vector to the desired number of classes. In total the network
# has 23 convolutional layers.
# To allow a seamless tiling of the output segmentation map (see Figure 2), it
# is important to select the input tile size such that all 2x2 max-pooling operations
# are applied to a layer with an even x- and y-size.

# The reduced output size within a single tile (e.g., 388x388 for a 572x572 input) ensur
# avoiding incomplete or invalid segmentations near the borders.
```

```
#pytorch libraries
import torch
import torch.nn as nn
import torch.nn.functional as F #for ReLu
import torchvision.transforms.functional as Trans
from torchinfo import summary

class UNet(nn.Module):
    def __init__(self, input_number, output_number):
        super(UNet, self).__init__()
        self.input_number = input_number
        self.output_number = output_number
```

```
        # Encoder
        # input: 4d tensor: batch_size x input_number x 572x572 --> input number=1 for g
        # assuming 572x572 image
        self.conv1 = nn.Conv2d(self.input_number, 64, kernel_size=3, padding=0) # 64x570
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, padding=0) # 64x568x568
        self.maxPool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 64x284x284

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=0) # 128x282x282
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=0) # 128x280x280
        self.maxPool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 128x140x140

        self.conv5 = nn.Conv2d(128,256, kernel_size=3, padding=0) # 256x138x138
        self.conv6 = nn.Conv2d(256, 256, kernel_size=3, padding=0) # 256x136x136
        self.maxPool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 256x68x68

        self.conv7 = nn.Conv2d(256, 512, kernel_size=3, padding=0) # 512x66x66
        self.conv8 = nn.Conv2d(512, 512, kernel_size=3, padding=0) # 512x64x64
        self.maxPool4 = nn.MaxPool2d(kernel_size=2, stride=2) # 512x32x32

        self.conv9 = nn.Conv2d(512, 1024, kernel_size=3, padding=0) # 1024x30x30
        self.conv10 = nn.Conv2d(1024, 1024, kernel_size=3, padding=0) # 1024x28x28

        # Decoder
        # Upsampling by a factor of 2, --> stride=2, kernel_size=2
        # 2x2 up convolution halves the feature channels

        self.upconv1 = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2) # 512x56x5
        self.conv1b = nn.Conv2d(1024, 512, kernel_size=3, padding=0) # 512x54x54 other 5
        self.conv2b = nn.Conv2d(512, 512, kernel_size=3, padding=0) # 512x52x52

        self.upconv2 = nn.ConvTranspose2d(512, 256, 2, 2) # 512x104x104
        self.conv3b = nn.Conv2d(512, 256, 3, padding=0) #256x102x102
        self.conv4b = nn.Conv2d(256, 256, 3, padding=0) # 256x100x100

        self.upconv3 = nn.ConvTranspose2d(256, 128, 2,2,) #256x200x200
        self.conv5b = nn.Conv2d(256, 128, 3, padding=0) #128x198x198
        self.conv6b = nn.Conv2d(128, 128, 3, padding=0) #128x196x196

        self.upconv4 = nn.ConvTranspose2d(128, 64, 2, 2) #128x392x392
        self.conv7b = nn.Conv2d(128, 64, 3, padding=0) #64x390x390
        self.conv8b = nn.Conv2d(64,64,3, padding=0) # 64x388x388
        self.final_conv = nn.Conv2d(64, self.output_number, kernel_size=1, padding=0) #2

    def cropConcat(self, encoder, decoder):
        # crops the encoder tensor and concatenate its with the decoder tensor
        _,_,H,W = decoder.shape
        cropped_enc = Trans.center_crop(encoder, [H,W]) # crops the encoder tensor in th
        return torch.cat((cropped_enc, decoder), dim=1) # concatenates at the feature di

    def forward(self, x):
        # Encoder
        x = F.relu(self.conv1(x))
        x1 = F.relu(self.conv2(x))
        x = self.maxPool1(x1)
```

```
        x = F.relu(self.conv3(x))
        x2 = F.relu(self.conv4(x))
        x = self.maxPool2(x2)

        x = F.relu(self.conv5(x))
        x3 = F.relu(self.conv6(x))
        x = self.maxPool3(x3)

        x = F.relu(self.conv7(x))
        x4 = F.relu(self.conv8(x))
        x = self.maxPool4(x4)

        x = F.relu(self.conv9(x))
        x = F.relu(self.conv10(x))

        # Decoder
        x = self.upconv1(x)  # size 512x56x56

        x = self.cropConcat(x4, x) # concatination1 size 1024x56x56
        x = F.relu(self.conv1b(x))
        x = F.relu(self.conv2b(x))
        x = self.upconv2(x)

        x = self.cropConcat(x3,x)
        x = F.relu(self.conv3b(x))
        x = F.relu(self.conv4b(x))
        x = self.upconv3(x)

        x = self.cropConcat(x2,x)
        x = F.relu(self.conv5b(x))
        x = F.relu(self.conv6b(x))
        x = self.upconv4(x)

        x = self.cropConcat(x1,x)
        x = F.relu(self.conv7b(x))
        x = F.relu(self.conv8b(x))
        x = self.final_conv(x)
        #x = F.softmax(x,1)
        return x


model = UNet(1,1)
summary(model, input_size=(1, 1, 572, 572))  # Example input size
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
UNet                                     [1, 1, 388, 388]          --
├─Conv2d: 1-1                            [1, 64, 570, 570]         640
├─Conv2d: 1-2                            [1, 64, 568, 568]         36,928
├─MaxPool2d: 1-3                         [1, 64, 284, 284]         --
├─Conv2d: 1-4                            [1, 128, 282, 282]        73,856
├─Conv2d: 1-5                            [1, 128, 280, 280]        147,584
```

```
├─MaxPool2d: 1-6                         [1, 128, 140, 140]        --
├─Conv2d: 1-7                            [1, 256, 138, 138]        295,168
├─Conv2d: 1-8                            [1, 256, 136, 136]        590,080
├─MaxPool2d: 1-9                         [1, 256, 68, 68]          --
├─Conv2d: 1-10                           [1, 512, 66, 66]          1,180,160
├─Conv2d: 1-11                           [1, 512, 64, 64]          2,359,808
├─MaxPool2d: 1-12                        [1, 512, 32, 32]          --
├─Conv2d: 1-13                           [1, 1024, 30, 30]         4,719,616
├─Conv2d: 1-14                           [1, 1024, 28, 28]         9,438,208
├─ConvTranspose2d: 1-15                  [1, 512, 56, 56]          2,097,664
├─Conv2d: 1-16                           [1, 512, 54, 54]          4,719,104
├─Conv2d: 1-17                           [1, 512, 52, 52]          2,359,808
├─ConvTranspose2d: 1-18                  [1, 256, 104, 104]        524,544
├─Conv2d: 1-19                           [1, 256, 102, 102]        1,179,904
├─Conv2d: 1-20                           [1, 256, 100, 100]        590,080
├─ConvTranspose2d: 1-21                  [1, 128, 200, 200]        131,200
├─Conv2d: 1-22                           [1, 128, 198, 198]        295,040
├─Conv2d: 1-23                           [1, 128, 196, 196]        147,584
├─ConvTranspose2d: 1-24                  [1, 64, 392, 392]         32,832
├─Conv2d: 1-25                           [1, 64, 390, 390]         73,792
├─Conv2d: 1-26                           [1, 64, 388, 388]         36,928
├─Conv2d: 1-27                           [1, 1, 388, 388]          65
==========================================================================================
Total params: 31,030,593
Trainable params: 31,030,593
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 167.45
==========================================================================================
Input size (MB): 1.31
Forward/backward pass size (MB): 1073.62
Params size (MB): 124.12
Estimated Total Size (MB): 1199.05
==========================================================================================
```

```
# Script for visualizing and testing the U-net model
# Explanation of code in visualization_ReadMe.md

from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import random
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor, transforms
import torchvision.models as models
import torchvision.transforms.functional as Trans

# Count images in TIF file
def countImages(image):
    # Count images
    image_count = 0
    try:
        while True:
            image.seek(image_count)
```

```
            image_count +=1
    except EOFError:
        pass

    print(f"Number of pages: {image_count}\n")
    return image_count

# Display all Images of the dataset
def displayImages(image, title):
    plt.figure(figsize=(20,20))
    for i in range(countImages(image)):
        try:
            image.seek(i)
            plt.subplot(6,5, i+1)
            plt.imshow(image)
            plt.axis("off")
            plt.title(f"Page {i+1}")
        except EOFError:
            break

    plt.suptitle(title, fontsize=16)
    plt.tight_layout()
    plt.show(block=False)

# Prediction function for a single image
def predict_single(image, model):
    model.eval()
    image = image.unsqueeze(0).to(device)  # Add batch dimension and move to device
    with torch.no_grad():
        pred = model(image)
    return pred


# function only for testing purposes of visualization
def evaluate_test(prediction, expected):
    probabilities = torch.sigmoid(prediction)
    probabilities = prediction
    predicted_segmentation = (probabilities > 0.5).int()  #convert to binary mask (1,1

    _,H,W = predicted_segmentation.shape
    H=388
    W=388
    expected_segmentation = Trans.center_crop(expected, [H,W]) # crop the expected lab
    predicted_segmentation = Trans.center_crop(predicted_segmentation, [H,W]) # crop t

    diff = (expected_segmentation != predicted_segmentation).int()
    return predicted_segmentation, expected_segmentation, diff  #return both binary ma

# correct evaluation function
def evaluate(prediction, expected):
    probabilities = torch.sigmoid(prediction)
    predicted_segmentation = (probabilities > 0.5).int()  #convert to binary mask (1,1

    _,H,W = predicted_segmentation.shape
```

```
    expected_segmentation = Trans.center_crop(expected, [H,W]) # crop the expected lab

    diff = (expected_segmentation != predicted_segmentation).int() # difference mask
    return predicted_segmentation, expected_segmentation, diff  #return both binary ma

# selecting device and loading model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#model = torch.load('u_net.pth', weights_only=False)
#model = model.to(device)



# ----------------------------------------------------------------------
# Displaying the complete  train/test dataset
#Training images
train_image = Image.open("ISBI-2012-challenge/train-mirror.tif")
#displayImages(train_image, "Training images")
# Training labels
train_label = Image.open("ISBI-2012-challenge/train-labels.tif")
#displayImages(train_label, "Training labels")

# #Test images
test_image = Image.open("ISBI-2012-challenge/test-mirror.tif")
# displayImages(test_image, "Test images")
# Training labels
test_label = Image.open("ISBI-2012-challenge/test-labels.tif")
# displayImages(test_label, "Test labels")
#input("press enter for close")

#---------------------------------------------------------------------



# ---------------------------------------------------------------------
# Visualization
# What we want to display: Image, Prediction; Groundtruth, Difference
# Image size = 512x512,
# Prediction size = 388x388
# select a random image of test set
randomID = random.randint(1, countImages(test_image))
test_image.seek(randomID)
test_label.seek(randomID)

plt.figure(figsize=(30,30))

# Test image
plt.subplot(2,2, 1)
plt.imshow(test_image)
plt.title(f"Test image")
plt.axis("off")


# Transform the test image/label to tensor
transform = transforms.Compose([transforms.ToTensor()]) # convert PIL image to (C,H,W)
```

```
image_tensor = transform(test_image)
label_tensor = transform(test_label)
#image_tensor = image_tensor.unsqueeze(0) #  Shape (1,C,H,W)
#torch.set_printoptions(threshold=torch.inf)
print(f"Tensor size image_tensor: {image_tensor.shape}") # (1,1,512,512)
print(image_tensor)
print(f"Tensor size label tensor: {label_tensor.shape}") # (1,1,512,512)
#print(label_tensor)

# Run the U-net for one image
#prediction = predict_single(test_image, model)

# --> prediction_mask, label_mask, diff = evaluate(prediction, label_tensor)
prediction_mask, label_mask, diff = evaluate(image_tensor, label_tensor)
print(f"Tensor size prediction mask: {prediction_mask.shape}")
print(f"Tensor size label mask: {label_mask.shape}")
prediction_mask.squeeze_(0)
label_mask.squeeze_(0)
diff.squeeze_(0)
plt.subplot(2,2,2)
plt.imshow(prediction_mask, cmap='viridis')
plt.title(f"Prediction")
cbar = plt.colorbar()
cbar.set_label("0: membrane, 1: membrane")
plt.axis("off")
plt.subplot(2,2,3)
plt.imshow(label_mask, cmap='viridis')
cbar = plt.colorbar()
cbar.set_label("0: membrane, 1: cell")
plt.title(f"GT")
plt.axis("off")
plt.subplot(2,2,4)
plt.imshow(diff, cmap='viridis')
cbar = plt.colorbar()
cbar.set_label("0: correct, 1: wrong")
plt.title(f"Difference")
plt.axis("off")
# add colorbar

plt.show()
```

```
Number of pages: 30

Tensor size image_tensor: torch.Size([3, 572, 572])
tensor([[[0.4314, 0.2745, 0.1608,  ..., 0.5922, 0.5569, 0.4667],
         [0.4235, 0.3216, 0.2078,  ..., 0.5059, 0.3922, 0.2627],
         [0.4863, 0.3529, 0.2196,  ..., 0.5176, 0.4431, 0.3569],
         ...,
         [0.1843, 0.1529, 0.2078,  ..., 0.4314, 0.4157, 0.4314],
         [0.1804, 0.1294, 0.2549,  ..., 0.5882, 0.5020, 0.5490],
         [0.1608, 0.0745, 0.2471,  ..., 0.6706, 0.6039, 0.5961]],

        [[0.4314, 0.2745, 0.1608,  ..., 0.5922, 0.5569, 0.4667],
         [0.4235, 0.3216, 0.2078,  ..., 0.5059, 0.3922, 0.2627],
         [0.4863, 0.3529, 0.2196,  ..., 0.5176, 0.4431, 0.3569],
```

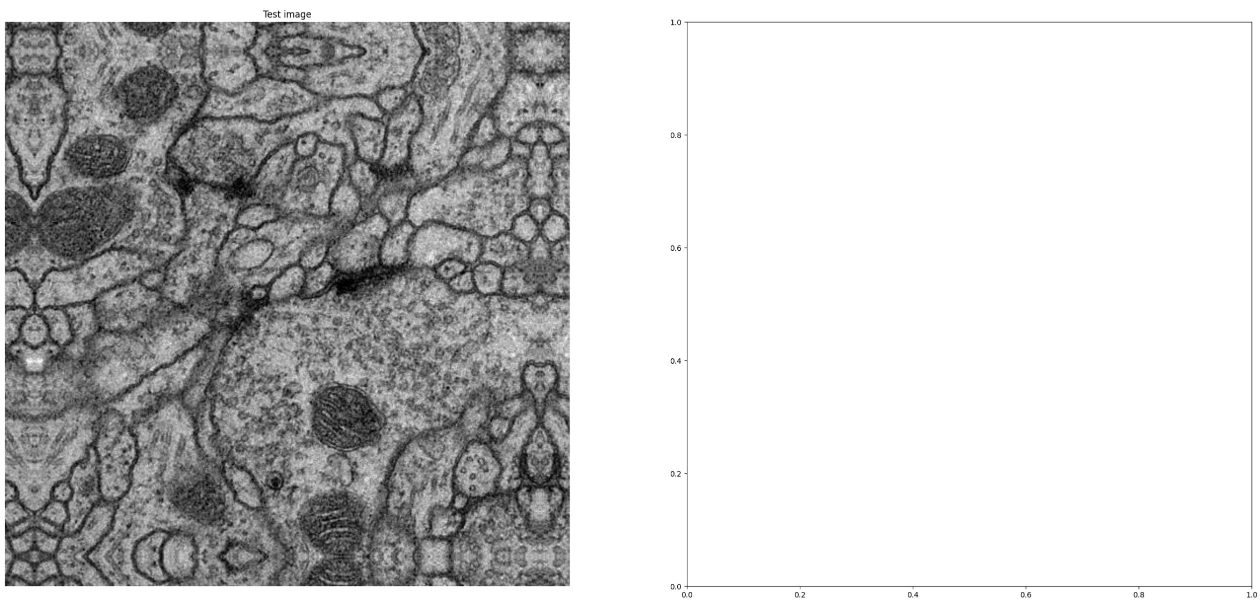
```
         ...,
         [0.1843, 0.1529, 0.2078,  ..., 0.4314, 0.4157, 0.4314],
         [0.1804, 0.1294, 0.2549,  ..., 0.5882, 0.5020, 0.5490],
         [0.1608, 0.0745, 0.2471,  ..., 0.6706, 0.6039, 0.5961]],

        [[0.4314, 0.2745, 0.1608,  ..., 0.5922, 0.5569, 0.4667],
         [0.4235, 0.3216, 0.2078,  ..., 0.5059, 0.3922, 0.2627],
         [0.4863, 0.3529, 0.2196,  ..., 0.5176, 0.4431, 0.3569],
         ...,
         [0.1843, 0.1529, 0.2078,  ..., 0.4314, 0.4157, 0.4314],
         [0.1804, 0.1294, 0.2549,  ..., 0.5882, 0.5020, 0.5490],
         [0.1608, 0.0745, 0.2471,  ..., 0.6706, 0.6039, 0.5961]]])
Tensor size label tensor: torch.Size([1, 512, 512])
Tensor size prediction mask: torch.Size([3, 572, 572])
Tensor size label mask: torch.Size([1, 572, 572])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-2bfce6c38239> in <cell line: 0>()
    149 diff.squeeze_(0)
    150 plt.subplot(2,2,2)
--> 151 plt.imshow(prediction_mask, cmap='viridis')
    152 plt.title(f"Prediction")
    153 cbar = plt.colorbar()
```

4 frames

```
/usr/local/lib/python3.11/dist-packages/matplotlib/image.py in
_normalize_image_array(A)
    641             A = A.squeeze(-1)  # If just (M, N, 1), assume scalar and
apply colormap.
    642         if not (A.ndim == 2 or A.ndim == 3 and A.shape[-1] in [3, 4]):
--> 643             raise TypeError(f"Invalid shape {A.shape} for image data")
    644         if A.ndim == 3:
    645             # If the input data has values outside the valid range (after

TypeError: Invalid shape (3, 572, 572) for image data
```



Next steps: Explain error

```
import torch
from torch import optim, nn
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms
import torchvision.transforms.functional as Trans
from PIL import Image
import matplotlib.pyplot as plt

torch.cuda.empty_cache()

#Loading data
#https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
class Dataset(Dataset):
  def __init__(self, image_path, label_path, transform=None):
    self.images = Image.open(image_path)
    self.labels = Image.open(label_path)
    self.transform = transform

  def __len__(self):
    return self.images.n_frames

  def __getitem__(self, idx):
    #find specific frame
    self.images.seek(idx)
    self.labels.seek(idx)
    #grayscale conversion (if necessary)
    image = self.images.convert("L")
    label = self.labels.convert("L")
    if self.transform:
      image = self.transform(image)
      label = self.transform(label)
    return image, label

transform = transforms.Compose([transforms.ToTensor()])

#original
#train_dataset = Dataset("ISBI-2012-challenge/train-volume.tif", "ISBI-2012-challenge/tr
#test_dataset = Dataset("ISBI-2012-challenge/test-volume.tif", "ISBI-2012-challenge/test

#mirrored
train_dataset = Dataset("ISBI-2012-challenge/train-mirror.tif", "ISBI-2012-challenge/tra
test_dataset = Dataset("ISBI-2012-challenge/test-mirror.tif", "ISBI-2012-challenge/test-


train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)
```

```
#print(train_dataset.images.n_frames)

#print(train_dataset.shape)

#Training
#https://pytorch.org/tutorials/beginner/introyt/trainingyt.html
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = UNet(1,1).to(device)
optimizer= optim.Adam(model.parameters(),0.001)
#optimizer = optim.SGD(model.parameters(),0.001,momentum=0.99)
#criterion = nn.CrossEntropyLoss() #for some reason doesn't work
criterion = nn.BCEWithLogitsLoss()


def train(model, dataloader, criterion, optimizer, nrOfEpochs):
  model.train()
  for i in range(nrOfEpochs):
    running_loss = 0.0
    avgAcc=0.0
    avgPrec=0.0
    avgRec=0.0
    avgF1=0.0
    for images, labels in dataloader:
      images, labels = images.to(device), labels.to(device);
      #print(images.shape)
      optimizer.zero_grad()
      outputs=model(images)
      #outputs = (outputs > 0.5).float()
      labels = Trans.center_crop(labels, [388,388])
      #assert outputs.shape == labels.shape, f"Shape mismatch: {outputs.shape} vs {label
      loss = criterion(outputs,labels)
      loss.backward()
      optimizer.step()
      running_loss += loss.item()
      if(i==nrOfEpochs-1):
        for j in range(dataloader.batch_size):
          outputs = (outputs > 0.5).int()
          labels = labels.int()
          TP = ((labels[j])*(outputs[j])).sum()
          TN = ((1-labels[j])*(1-outputs[j])).sum()
          FP = ((1-labels[j])*(outputs[j])).sum()
          FN = ((labels[j])*(1-outputs[j])).sum()

          accuracy = (TP+TN)/(TP+TN+FP+FN)
          precision = TP/(TP+FP)
          recall = TP/(TP+FN)
          f1 = 2*(precision*recall)/(precision+recall)
          '''
          print(f"Accuracy:{accuracy}")
          print(f"Precision:{precision}")
          print(f"Recall:{recall}")
          print(f"F1 Score:{f1}\n")
          '''
```

```
            avgAcc+=accuracy.cpu().item()
            avgPrec+=precision.cpu().item()
            avgRec+=recall.cpu().item()
            avgF1+=f1.cpu().item()

      '''
      #for visualization of the training data
      for j in range(dataloader.batch_size):
          plt.figure()
          plt.subplot(2,2,1)
          plt.imshow(images[j].cpu().numpy().squeeze(), cmap='viridis')
          plt.subplot(2,2,2)
          plt.imshow(labels[j].cpu().numpy().squeeze(), cmap='viridis')
      '''
      train_loss = running_loss/len(dataloader)
      print(f"Epoch {i+1}/{nrOfEpochs}\nLoss:{train_loss}")
    return train_loss, avgAcc/len(dataloader), avgPrec/len(dataloader), avgRec/len(dataloa


  #Evaluation
  def test(model, dataloader, criterion):
    model.eval()
    running_loss = 0.0
    avgAcc=0.0
    avgPrec=0.0
    avgRec=0.0
    avgF1=0.0
    plotImage=0
    with torch.no_grad():
      for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device);
        outputs = model(images)
        labels = Trans.center_crop(labels, [388,388])
        loss = criterion(outputs,labels)
        running_loss+=loss.item()
        outputs = (outputs > 0.5).int()
        #visualisation and evaluation
        for j in range(dataloader.batch_size):
          if(plotImage==6): #for picking single image to plot
            plt.figure()
            plt.subplot(2,2,1)
            plt.title(f"Image")
            plt.imshow(images[j].cpu().numpy().squeeze(), cmap='viridis')
            plt.subplot(2,2,2)
            plt.title(f"Ground Truth")
            plt.imshow(labels[j].cpu().numpy().squeeze(), cmap='viridis')
            plt.subplot(2,2,3)
            plt.title(f"Prediction")
            #outputs[j]=(outputs[j]>0.5).int()
            plt.imshow(outputs[j].cpu().detach().numpy().squeeze(), cmap='viridis')
            plt.subplot(2,2,4)
            plt.title(f"Difference")
            diff=(outputs[j]!=labels[j]).int()
            plt.imshow(diff.cpu().detach().numpy().squeeze(),cmap='viridis')
```

```
            plt.tight_layout()
            plt.show()

          #Output image to binary values
          #outputs[j] = (outputs > 0.5).float()
          labels = labels.int()
          TP = ((labels[j])*(outputs[j])).sum()
          TN = ((1-labels[j])*(1-outputs[j])).sum()
          FP = ((1-labels[j])*(outputs[j])).sum()
          FN = ((labels[j])*(1-outputs[j])).sum()

          accuracy = (TP+TN)/(TP+TN+FP+FN)
          precision = TP/(TP+FP)
          recall = TP/(TP+FN)
          f1 = 2*(precision*recall)/(precision+recall)
          '''
          print(f"Accuracy:{accuracy}")
          print(f"Precision:{precision}")
          print(f"Recall:{recall}")
          print(f"F1 Score:{f1}\n")
          '''
          avgAcc+=accuracy.cpu().item()
          avgPrec+=precision.cpu().item()
          avgRec+=recall.cpu().item()
          avgF1+=f1.cpu().item()

          plotImage+=1

    print(f"Avg Accuracy:{avgAcc/len(dataloader)}")
    print(f"Avg Precision:{avgPrec/len(dataloader)}")
    print(f"Avg Recall:{avgRec/len(dataloader)}")
    print(f"Avg F1 Score:{avgF1/len(dataloader)}\n")

    test_loss = running_loss/len(dataloader)
    print(f"Test loss:{test_loss}")
    return test_loss, avgAcc/len(dataloader), avgPrec/len(dataloader), avgRec/len(dataloac

#10x number of epochs
nrEpx10 = 20

x = [0]*nrEpx10
trainLoss = [0]*nrEpx10
trainAcc = [0]*nrEpx10
trainPrec = [0]*nrEpx10
trainRec = [0]*nrEpx10
trainF1 = [0]*nrEpx10
testLoss = [0]*nrEpx10
testAcc = [0]*nrEpx10
testPrec = [0]*nrEpx10
testRec = [0]*nrEpx10
testF1 = [0]*nrEpx10


#torch.load('unet_50ep_lr0001_mirr.pth', map_location=torch.device('cpu'))
```

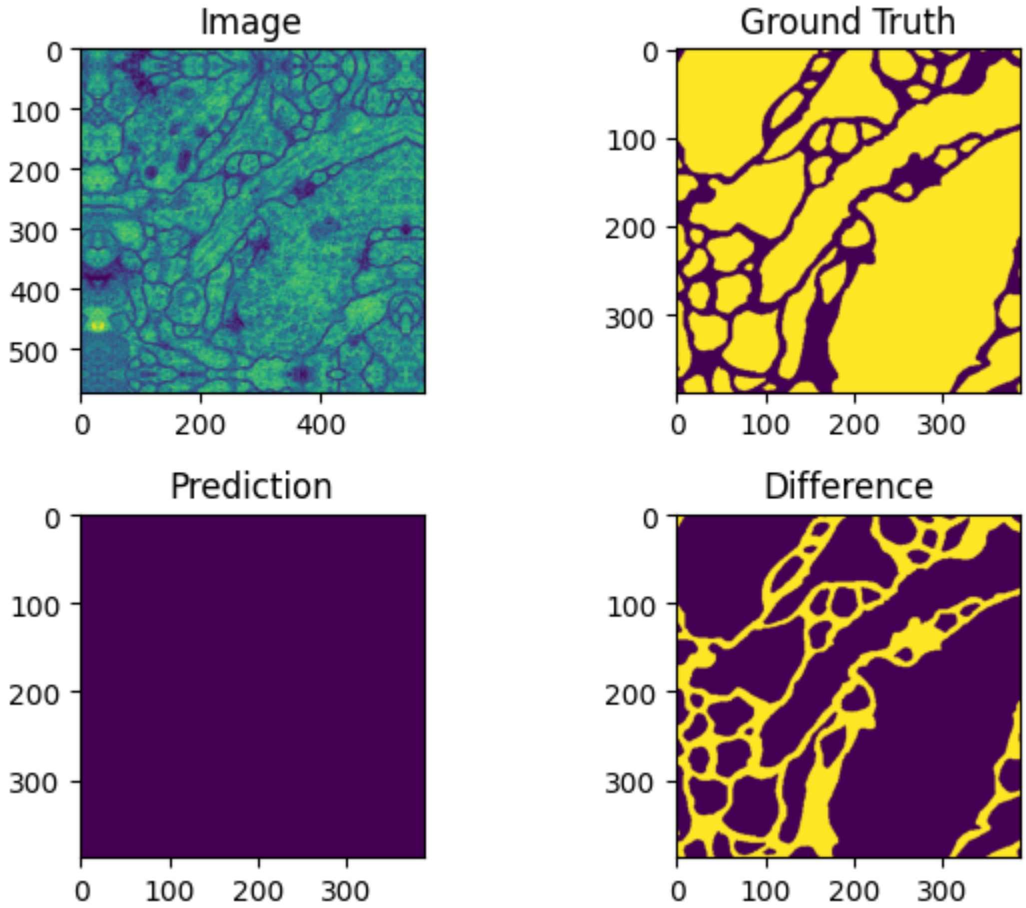
```
#Running code:
for i in range(nrEpx10):
  print(f"\nLoop {i+1}\n")
  x[i] = 10*(i+1)
  trainLoss[i], trainAcc[i], trainPrec[i], trainRec[i], trainF1[i] = train(model, train_
  testLoss[i], testAcc[i], testPrec[i], testRec[i], testF1[i] = test(model, test_loader,
  #update frame

#Plot acc, prec, rec, f1
fig = plt.figure()
#plt.title('Metrics over Epochs')
plt.subplot(2,3,1)
#plt.title(f"Loss")
plt.plot(x, trainLoss, "-o", label='Train Loss')
plt.plot(x, testLoss, "-o", label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.subplot(2,3,2)
#plt.title(f"Accuracy")
plt.plot(x, trainAcc, "-o", label='Train Accuracy')
plt.plot(x, testAcc, "-o", label='Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.subplot(2,3,3)
#plt.title(f"Precision")
plt.plot(x, trainPrec, "-o", label='Train Precision')
plt.plot(x, testPrec, "-o", label='Test Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.subplot(2,3,4)
#plt.title(f"Recall")
plt.plot(x, trainRec, "-o", label='Train Recall')
plt.plot(x, testRec, "-o", label='Test Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.subplot(2,3,5)
#plt.title(f"F1")
#plt.plot(x, trainF1, "-o", label='Train F1 Score')
#plt.plot(x, testF1, "-o", label='Test F1 Score')
plt.plot(x, trainF1, "-o", label='Train')
plt.plot(x, testF1, "-o", label='Test')
plt.xlabel('Epochs')
plt.ylabel('F1')
#plt.legend()
handles, labels = plt.gca().get_legend_handles_labels()
fig.legend(handles, labels, loc='lower right')
plt.tight_layout()
plt.show()


#Save model - change name to: unet _ nr of epochs _ learning rate 0.xxx as lrxxx _ which
#torch.save(model.state_dict(), 'unet_100ep_lr00001_mirr.pth')
```

```
Loop 1

Epoch 1/10
Loss:0.6694238523642222
Epoch 2/10
Loss:0.591159118215243
Epoch 3/10
Loss:0.528099466363589
Epoch 4/10
Loss:0.525952356060346
Epoch 5/10
Loss:0.5217139720916748
Epoch 6/10
Loss:0.5194161375363667
Epoch 7/10
Loss:0.489606378475825
Epoch 8/10
Loss:0.47040367821852364
Epoch 9/10
Loss:0.4394596715768178
Epoch 10/10
Loss:0.40733722150325774
```

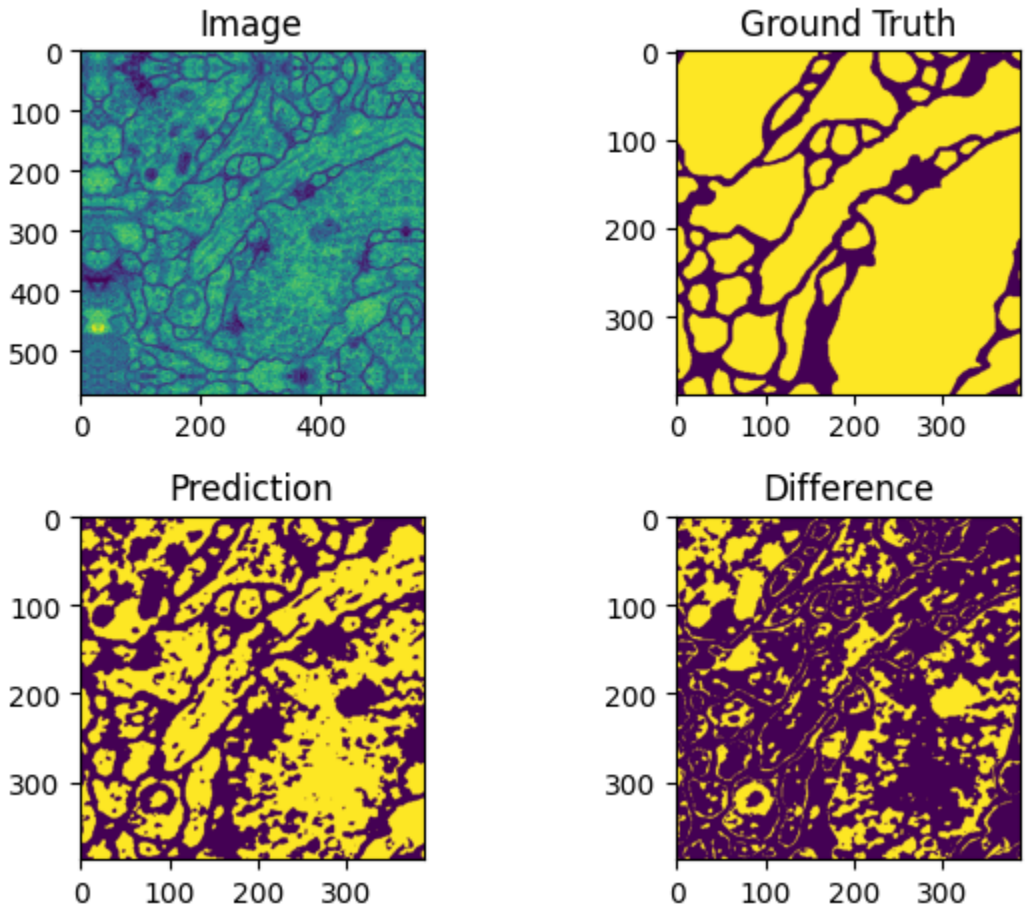



```
Avg Accuracy:0.7215602477391561
Avg Precision:0.7215602477391561
Avg Recall:1.0
Avg F1 Score:0.8374458173910777

Test loss:0.5187153120835623
```

```
Loop 2

Epoch 1/10
Loss:0.3851223429044088
Epoch 2/10
Loss:0.3774399360020955
Epoch 3/10
Loss:0.38439785738786064
Epoch 4/10
Loss:0.3774010419845581
Epoch 5/10
Loss:0.35593909124533335
Epoch 6/10
Loss:0.34559838275114696
Epoch 7/10
Loss:0.36820192337036134
Epoch 8/10
Loss:0.3587419251600901
Epoch 9/10
Loss:0.37042352159818015
Epoch 10/10
Loss:0.3790782610575358
```



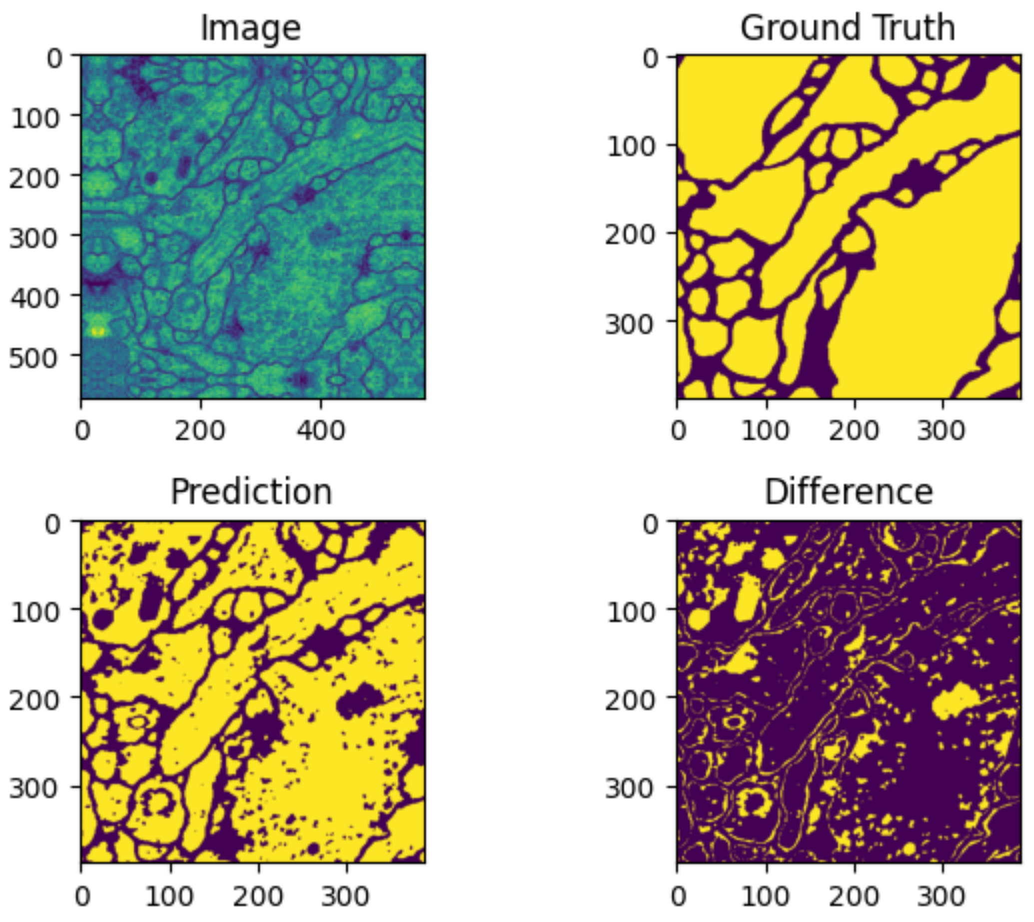

```
Avg Accuracy:0.776786188284556
Avg Precision:0.9115005950133006
Avg Recall:0.7718825697898865
Avg F1 Score:0.8300122777620952

Test loss:0.42017103532950084

Loop 3

Epoch 1/10
```

```
Loss:0.36321892539660133
Epoch 2/10
Loss:0.3592783729235331
Epoch 3/10
Loss:0.35193413893381753
Epoch 4/10
Loss:0.3519824892282486
Epoch 5/10
Loss:0.3519705325365067
Epoch 6/10
Loss:0.3483248939116796
Epoch 7/10
Loss:0.3453931788603465
Epoch 8/10
Loss:0.3425674001375834
Epoch 9/10
Loss:0.3407235950231552
Epoch 10/10
Loss:0.34316649635632834
```



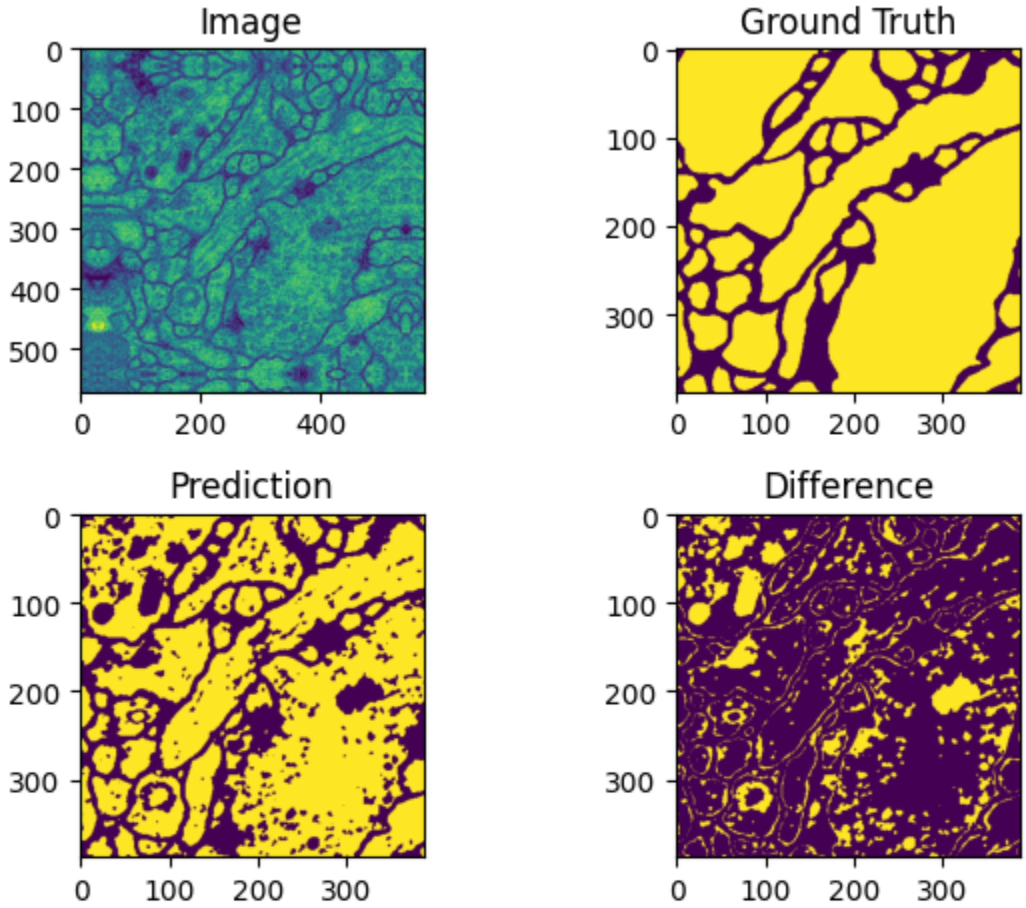

```
Avg Accuracy:0.7986314098040262
Avg Precision:0.8514258642991384
Avg Recall:0.8803501745065053
Avg F1 Score:0.8622707684834798

Test loss:0.4586367626984914

Loop 4

Epoch 1/10
Loss:0.3406229704618454
Epoch 2/10
```

```
Loss:0.3457259794076284
Epoch 3/10
Loss:0.33847936590512595
Epoch 4/10
Loss:0.3443219393491745
Epoch 5/10
Loss:0.34111566146214806
Epoch 6/10
Loss:0.3407439102729162
Epoch 7/10
Loss:0.345645668109258
Epoch 8/10
Loss:0.34453535278638203
Epoch 9/10
Loss:0.34273766775925957
Epoch 10/10
Loss:0.33886585334936775
```




```
Avg Accuracy:0.8007448554039002
Avg Precision:0.8723234335581461
Avg Recall:0.8552560230096181
Avg F1 Score:0.859752082824707

Test loss:0.4245744655529658

Loop 5

Epoch 1/10
Loss:0.3403441200653712
Epoch 2/10
Loss:0.34240088164806365
Epoch 3/10
Loss:0.33833454648653666
```

Start coding or generate with AI.