# Notebook: Comparing Segmentation models: Segment anything model (SAM) vs UNET on image segmentation

**In this notebook, we'll reproduce the SAM project by MetaAI, we will look into the model architectures of Segment anything model and UNET which has been mostly used image segmentation, we will later fine train SAM and UNET on a dataset of medical images.**


**Team member 1: Anupam Tiwari (ast9885)**

**Team member 2: Ankur Aggarwal (aa10336)**


NEW YORK UNIVERSITY

## Set-up environment

**We first install Transformers, Datasets and other nessasry libraries that our project is dependent on**

In [1]:

```
! pip install opencv-python pycocotools matplotlib onnxruntime onnx pycocotools supervision &> /dev/null
! pip install -q git+https://github.com/huggingface/transformers.git
! pip install -q datasets
! pip install git+https://github.com/facebookresearch/segment-anything.git &> /dev/null
! wget https://dl.fbaipublicfiles.com/segment_anything/sam_vit_b_01ec64.pth &> /dev/null
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 5.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 59.3 MB/s eta 0:00:00
  Building wheel for transformers (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 474.6/474.6 kB 10.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.5/110.5 kB 14.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 212.5/212.5 kB 24.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.3/134.3 kB 12.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 32.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.5/114.5 kB 13.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 26.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.6/149.6 kB 17.1 MB/s eta 0:00:00
```

**importing dependencies**

In [2]:

```
from pycocotools.coco import COCO
import os
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
import cv2
import json
from torch.utils.data import Dataset
import torch
import torchvision
from torch.utils.data import DataLoader
import albumentations as A
```

```
from albumentations.pytorch import ToTensorV2
from tqdm import tqdm
import torch.nn as nn
import torch.optim as optim
%matplotlib inline
```

# Load dataset

**Here we load a small dataset of 130 (image, ground truth mask) pairs. for SAM and UNET**

In [3]:

```
from datasets import load_dataset

dataset = load_dataset("nielsr/breast-cancer", split="train")
```

```
Downloading and preparing dataset None/None to /root/.cache/huggingface/datasets/nielsr__
_parquet/nielsr--breast-cancer-c16ee7932c43ffa3/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d3
1bd5b05b934492fdef7d8d6f236ec...
```

```
Dataset parquet downloaded and prepared to /root/.cache/huggingface/datasets/nielsr___par
quet/nielsr--breast-cancer-c16ee7932c43ffa3/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5
b05b934492fdef7d8d6f236ec. Subsequent calls will reuse this data.
```

**Lets look into the dataset: it has image and label pair, label is the mask**

In [4]:

```
dataset
```

Out[4]:

```
Dataset({
    features: ['image', 'label'],
    num_rows: 130
})
```

**example image:**

In [5]:

```
example = dataset[0]
image = example["image"]
image
```

Out[5]:

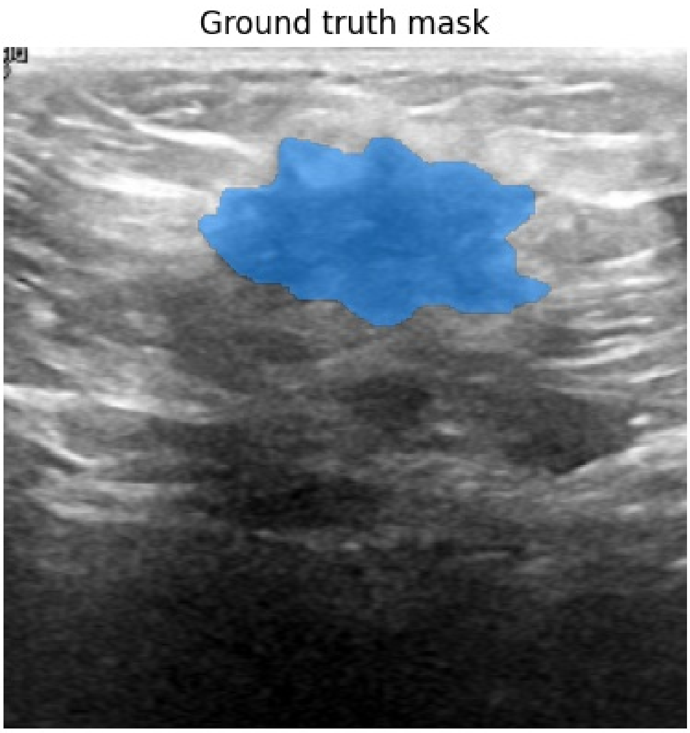
**IMAGE+MASK**

In [6]:

```
import matplotlib.pyplot as plt
import numpy as np

def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

fig, axes = plt.subplots()

axes.imshow(np.array(image))
ground_truth_seg = np.array(example["label"])
show_mask(ground_truth_seg, axes)
axes.title.set_text(f"Ground truth mask")
axes.axis("off")
```

Out[6]:

```
(-0.5, 255.5, 255.5, -0.5)
```



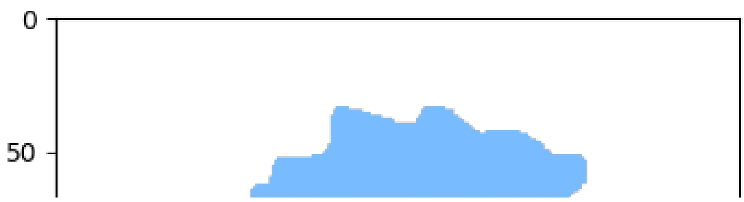

**Mask:**

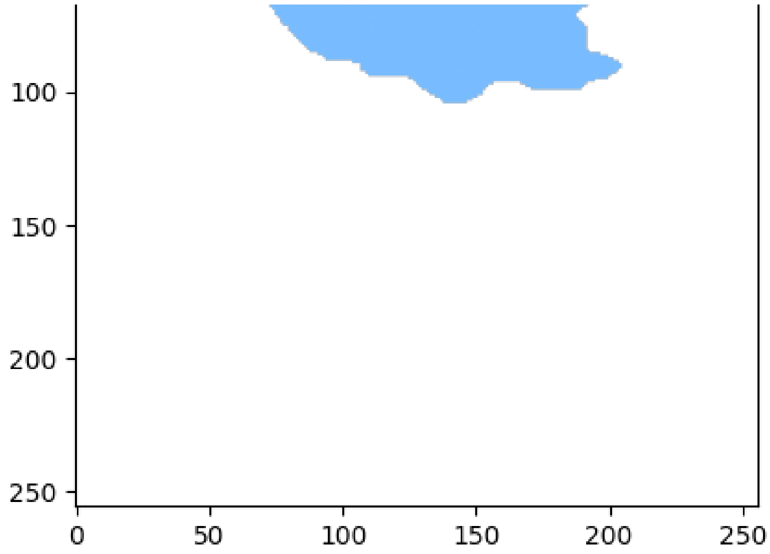
In [10]:

```
image = dataset[0]["image"]
mask = np.array(dataset[0]["label"])
h, w = mask.shape[-2:]
color = np.array([30/255, 144/255, 255/255, 0.6])
mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
plt.imshow(mask_image)
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x7f97849152d0>
```






**Now we will make two directories for Mask and Image, this is need for the UNET**

In [11]:

```
!mkdir mask
!mkdir img
```

```
mkdir: cannot create directory 'mask': File exists
mkdir: cannot create directory 'img': File exists
```

In [12]:

```
for i in range(130):
  image = np.array(dataset[i]["image"])
  mask = np.array(dataset[i]["label"])
  h, w = mask.shape[-2:]
  color = np.array([30/255, 144/255, 255/255, 0.6])
  mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
  file_name = str(i)
  cv2.imwrite('mask/'+file_name+".png", mask_image)
  cv2.imwrite('img/'+file_name+".png", image)
```

# Create PyTorch dataset

**Below we define a regular PyTorch dataset, which gives us examples of the data prepared in the format for the model. Each example consists of:**

**pixel values (which is the image prepared for the model) a prompt in the form of a bounding box a ground truth segmentation mask. The function below defines how to get a bounding box prompt based on the ground truth segmentation. This was taken from here.**

**Note that SAM is always trained using certain "prompts", which you could be bounding boxes, points, text, or rudimentary masks. The model is then trained to output the appropriate mask given the image + prompt.**

In [13]:

```
def get_bounding_box(ground_truth_map):
  # get bounding box from mask
  y_indices, x_indices = np.where(ground_truth_map > 0)
  x_min, x_max = np.min(x_indices), np.max(x_indices)
  y_min, y_max = np.min(y_indices), np.max(y_indices)
  # add perturbation to bounding box coordinates
  H, W = ground_truth_map.shape
  x_min = max(0, x_min - np.random.randint(0, 20))
  x_max = min(W, x_max + np.random.randint(0, 20))
  y_min = max(0, y_min - np.random.randint(0, 20))
  y_max = min(H, y_max + np.random.randint(0, 20))
  bbox = np.array([x_min, y_min, x_max, y_max])
```

```
    return bbox
```

In [15]:

```
bbox_coords = {}
for x in range(130):
  bbox_coords[str(x)] = get_bounding_box(np.array(dataset[x]['label']))
```

In [16]:

```
ground_truth_masks = {}
for k in bbox_coords.keys():
  gt_grayscale = np.array(dataset[int(k)]['label'])
  ground_truth_masks[k] = (gt_grayscale == 1)
ground_truth_masks['1']
```

Out[16]:

```
array([[False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False]])
```

In [17]:

```
from torch.utils.data import Dataset

class SAMDataset(Dataset):
  def __init__(self, dataset, processor):
    self.dataset = dataset
    self.processor = processor

  def __len__(self):
    return len(self.dataset)

  def __getitem__(self, idx):
    item = self.dataset[idx]
    image = item["image"]
    ground_truth_mask = np.array(item["label"])

    # get bounding box prompt
    prompt = get_bounding_box(ground_truth_mask)

    # prepare image and prompt for the model
    inputs = self.processor(image, input_boxes=[[prompt]], return_tensors="pt")

    # remove batch dimension which the processor adds by default
    inputs = {k:v.squeeze(0) for k,v in inputs.items()}

    # add ground truth segmentation
    inputs["ground_truth_mask"] = ground_truth_mask

    return inputs
```

In [18]:

```
from transformers import SamProcessor

processor = SamProcessor.from_pretrained("facebook/sam-vit-base")
```

In [19]:

```
train_dataset = SAMDataset(dataset=dataset, processor=processor)
```

In [20]:

```
example = train_dataset[0]
```

```
for k,v in example.items():
  print(k,v.shape)
```

```
pixel_values torch.Size([3, 1024, 1024])
original_sizes torch.Size([2])
reshaped_input_sizes torch.Size([2])
input_boxes torch.Size([1, 4])
ground_truth_mask (256, 256)
```

## Take a look at the images, the bounding box prompts and the ground truth segmentation masks

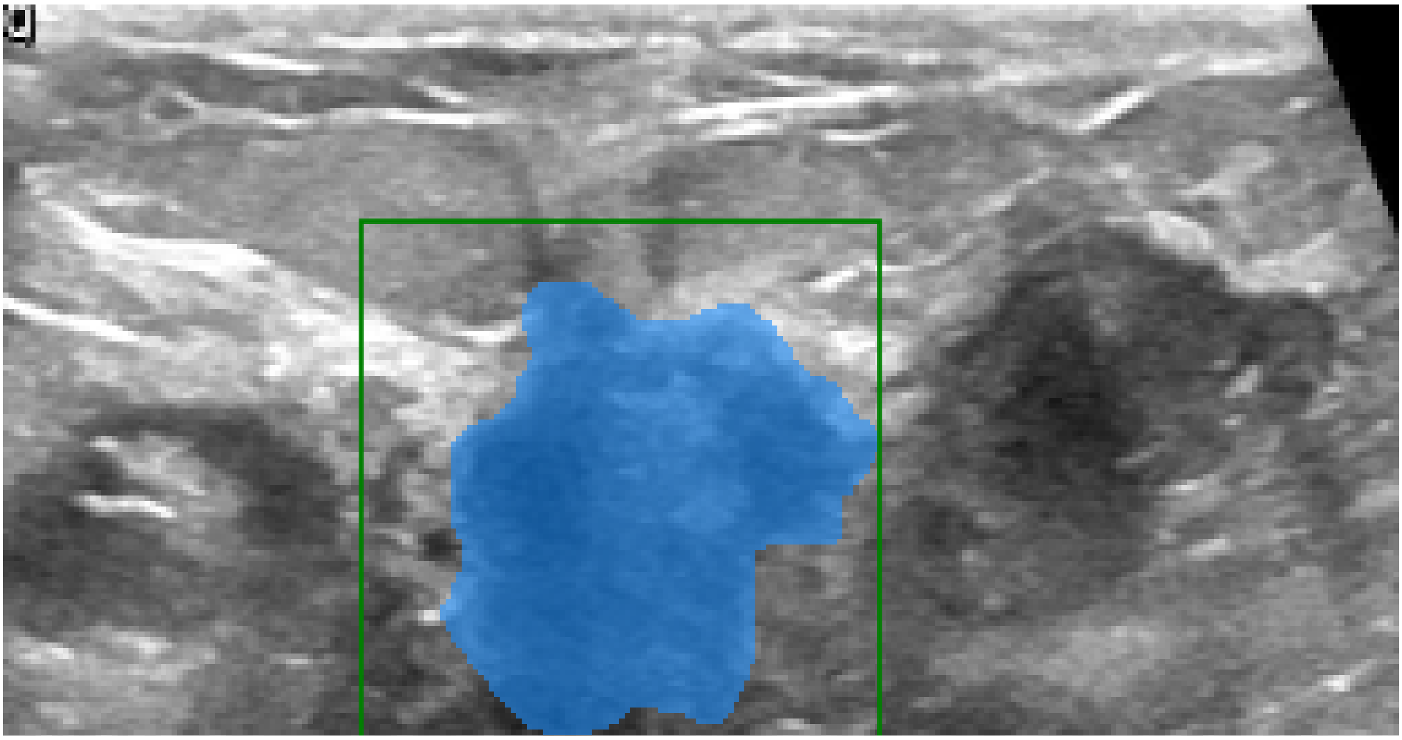
In [21]:

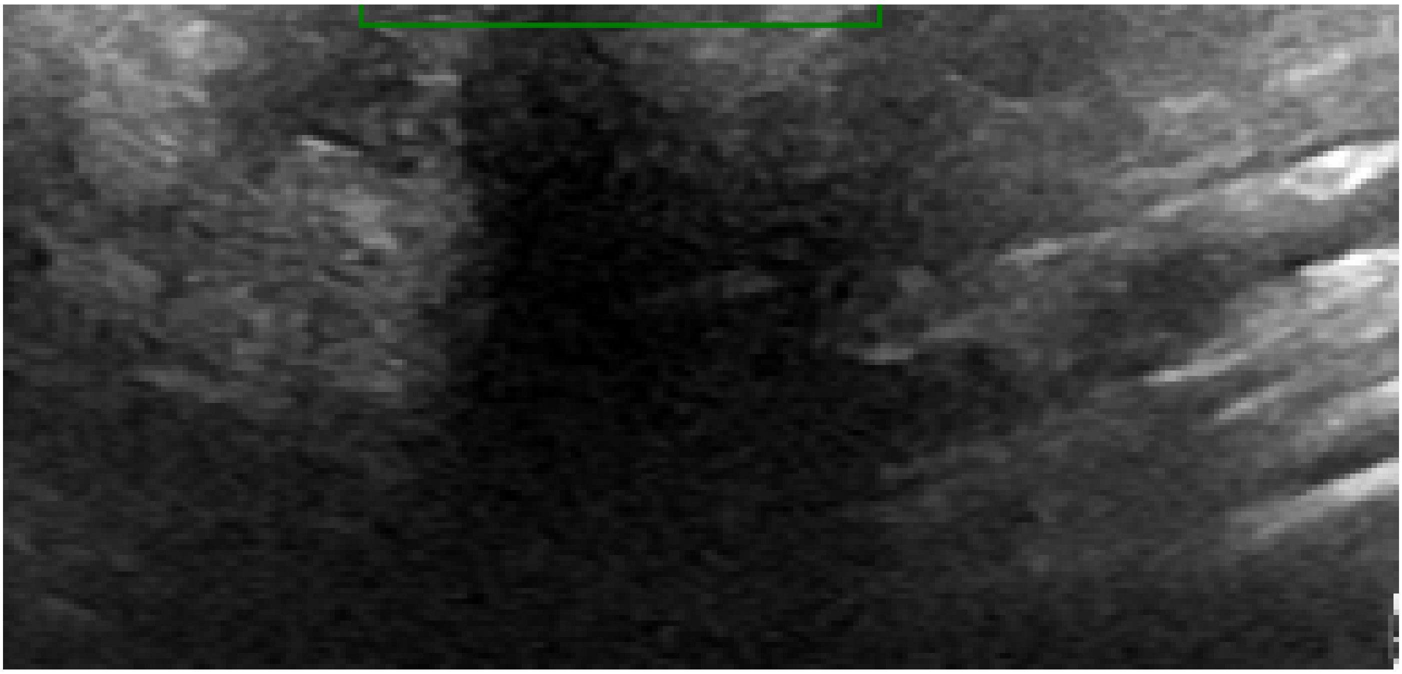
```
# Helper functions provided in https://github.com/facebookresearch/segment-anything/blob/
9e8f1309c94f1128a6e5c047a10fdcb02fc8d651/notebooks/predictor_example.ipynb
def show_mask(mask, ax, random_color=False):
    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_box(box, ax):
    print(box)
    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,0,0,0), l
w=2))
```

In [22]:

```
name = '1'
image = cv2.imread(f'img/{name}.png')

plt.figure(figsize=(10,10))
plt.imshow(image)
show_box(bbox_coords[name], plt.gca())
show_mask(ground_truth_masks[name], plt.gca())
plt.axis('off')
plt.show()
```

```
[ 65  39 160 137]
```





## Prepare Fine Tuning

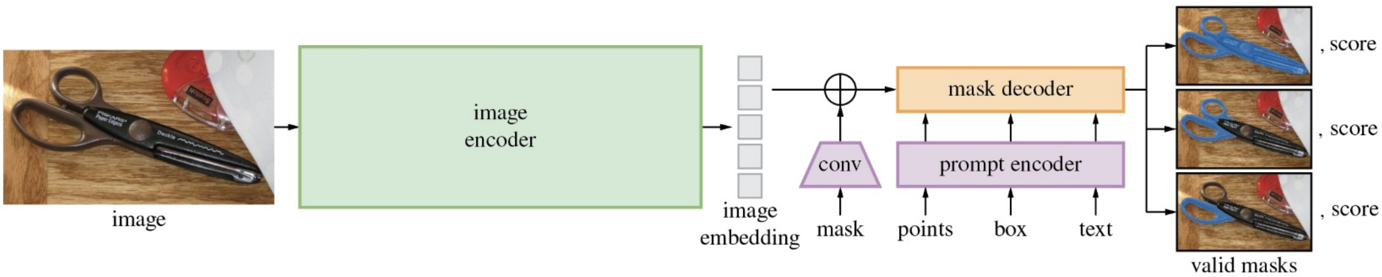
## Create PyTorch DataLoader

**Next we define a PyTorch Dataloader, which allows us to get batches from the dataset.**

In [23]:

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_dataset, batch_size=2, shuffle=True)
```

## Load Model




In [24]:

```
model_type = 'vit_b'
checkpoint = 'sam_vit_b_01ec64.pth'
device = 'cuda:0'
```

In [25]:

```
from transformers import SamModel

sam_model = SamModel.from_pretrained("facebook/sam-vit-base")

# make sure we only compute gradients for mask decoder
for name, param in sam_model.named_parameters():
  if name.startswith("vision_encoder") or name.startswith("prompt_encoder"):
    param.requires_grad_(False)
```

In [26]:

```
# Set up the optimizer, hyperparameter tuning will improve performance here
lr = 1e-4
wd = 0
optimizer = torch.optim.Adam(sam_model.mask_decoder.parameters(), lr=lr, weight_decay=wd
)

loss_fn = torch.nn.MSELoss()
# loss_fn = torch.nn.BCELoss()
keys = list(bbox_coords.keys())
```

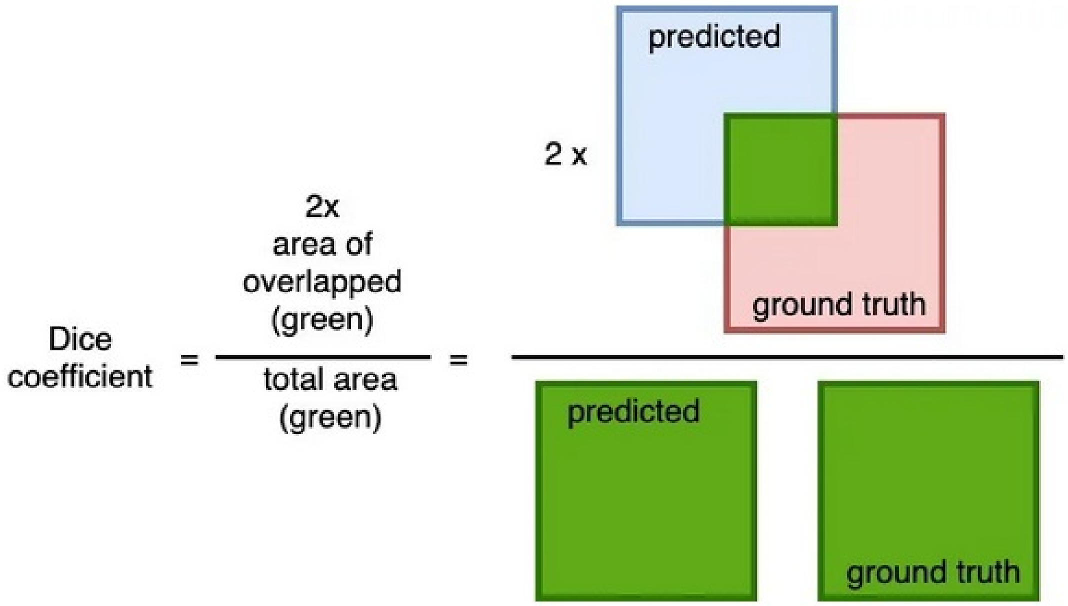
## Run fine tuning

**This is the main training loop.**

**Improvements to be made include batching and moving the computation of the image and prompt embeddings outside the loop since we are not tuning these parts of the model, this will speed up training as we should not recompute the embeddings during each epoch.**

**In a production implementation a better choice of optimiser/loss function will certainly help.**

**using monai for custom loss function, called dice loss**




In [28]:

```
!pip install monai
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Requirement already satisfied: monai in /usr/local/lib/python3.10/dist-packages (1.1.0)
Requirement already satisfied: torch>=1.8 in /usr/local/lib/python3.10/dist-packages (fro
m monai) (2.0.0+cu118)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fr
om monai) (1.22.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
torch>=1.8->monai) (3.12.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packag
es (from torch>=1.8->monai) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from tor
ch>=1.8->monai) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from
torch>=1.8->monai) (3.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from to
rch>=1.8->monai) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (
```

```
from torch>=1.8->monai) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from tri
ton==2.0.0->torch>=1.8->monai) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from trito
n==2.0.0->torch>=1.8->monai) (16.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
(from jinja2->torch>=1.8->monai) (2.1.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (f
rom sympy->torch>=1.8->monai) (1.3.0)
```

In [29]:

```python
from torch.optim import Adam
import monai

# Note: Hyperparameter tuning could improve performance here
optimizer = Adam(sam_model.mask_decoder.parameters(), lr=1e-5, weight_decay=0)

seg_loss = monai.losses.DiceCELoss(sigmoid=True, squared_pred=True, reduction='mean')
```

In [38]:

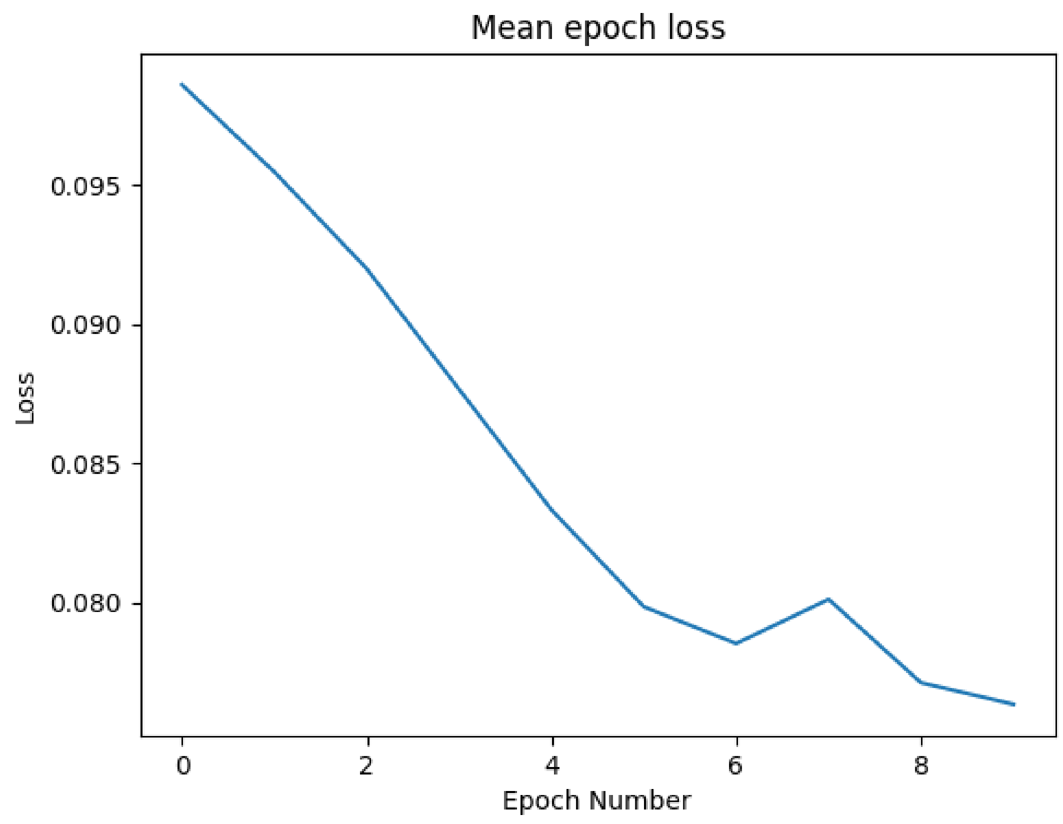
```python
from tqdm import tqdm
from statistics import mean
import torch
from torch.nn.functional import threshold, normalize

num_epochs = 10

device = "cuda" if torch.cuda.is_available() else "cpu"
sam_model.to(device)

sam_model.train()
losses = []
for epoch in range(num_epochs):
    epoch_losses = []
    for batch in tqdm(train_dataloader):
      # forward pass
      outputs = sam_model(pixel_values=batch["pixel_values"].to(device),
                          input_boxes=batch["input_boxes"].to(device),
                          multimask_output=False)

      # compute loss
      predicted_masks = outputs.pred_masks.squeeze(1)
      ground_truth_masks = batch["ground_truth_mask"].float().to(device)
      loss = seg_loss(predicted_masks, ground_truth_masks.unsqueeze(1))

      # backward pass (compute gradients of parameters w.r.t. loss)
      optimizer.zero_grad()
      loss.backward()

      # optimize
      optimizer.step()
      epoch_losses.append(loss.item())

    print(f'EPOCH: {epoch}')
    print(f'Mean loss: {mean(epoch_losses)}')
    losses.append(mean(epoch_losses))
```

```
100%|██████████| 65/65 [01:03<00:00,  1.02it/s]

EPOCH: 0
Mean loss: 0.09859017454660855

100%|██████████| 65/65 [01:02<00:00,  1.04it/s]

EPOCH: 1
Mean loss: 0.09546073858554546

100%|██████████| 65/65 [01:02<00:00,  1.03it/s]

EPOCH: 2
Mean loss: 0.09198274795825664
```

```
100%|██████████| 65/65 [01:02<00:00,  1.04it/s]

EPOCH: 3
Mean loss: 0.08765292396912208

100%|██████████| 65/65 [01:02<00:00,  1.03it/s]

EPOCH: 4
Mean loss: 0.08331799965638381

100%|██████████| 65/65 [01:02<00:00,  1.04it/s]

EPOCH: 5
Mean loss: 0.07983663311371436

100%|██████████| 65/65 [01:03<00:00,  1.03it/s]

EPOCH: 6
Mean loss: 0.07851512294549208

100%|██████████| 65/65 [01:02<00:00,  1.03it/s]

EPOCH: 7
Mean loss: 0.08010301269017733

100%|██████████| 65/65 [01:02<00:00,  1.04it/s]

EPOCH: 8
Mean loss: 0.07710946844174311

100%|██████████| 65/65 [01:02<00:00,  1.03it/s]

EPOCH: 9
Mean loss: 0.07633395791053772
```

In [39]:

```python
mean_losses = losses

plt.plot(list(range(len(mean_losses))), mean_losses)
plt.title('Mean epoch loss')
plt.xlabel('Epoch Number')
plt.ylabel('Loss')

plt.show()
```



## We can compare our tuned model to the original model

In [40]:

```
import numpy as np
from PIL import Image

# let's take a random training example
idx = 10

# load image
image = dataset[idx]["image"]
image
```

Out[40]:


In [41]:

```
# get box prompt based on ground truth segmentation map
ground_truth_mask = np.array(dataset[idx]["label"])
prompt = get_bounding_box(ground_truth_mask)

# prepare image + box prompt for the model
inputs = processor(image, input_boxes=[[[prompt]]], return_tensors="pt").to(device)
for k,v in inputs.items():
  print(k,v.shape)
```

```
pixel_values torch.Size([1, 3, 1024, 1024])
original_sizes torch.Size([1, 2])
reshaped_input_sizes torch.Size([1, 2])
input_boxes torch.Size([1, 1, 4])
```

In [42]:

```
sam_model.eval()

# forward pass
with torch.no_grad():
  outputs = sam_model(**inputs, multimask_output=False)
```

In [43]:

```
# apply sigmoid
medsam_seg_prob = torch.sigmoid(outputs.pred_masks.squeeze(1))
# convert soft mask to hard mask
medsam_seg_prob = medsam_seg_prob.cpu().numpy().squeeze()
medsam_seg = (medsam_seg_prob > 0.5).astype(np.uint8)
```

In [44]:

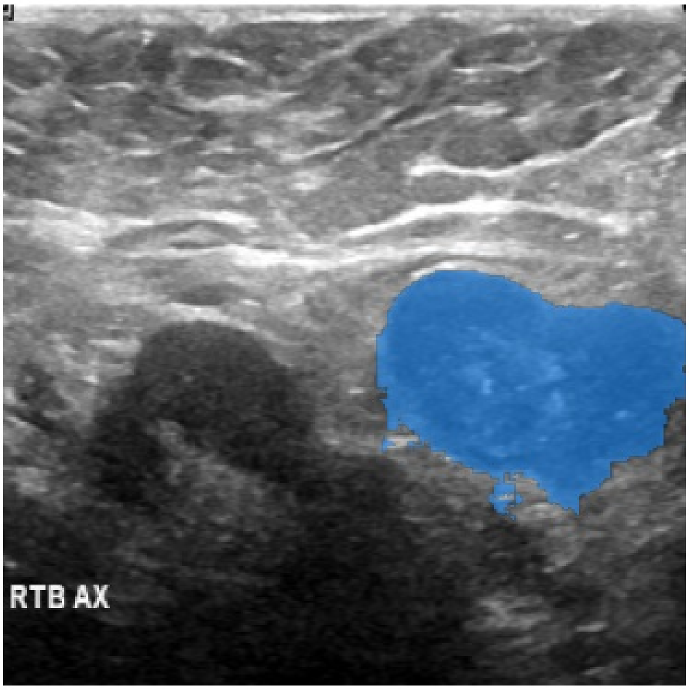
```
def show_mask(mask, ax, random_color=False):
    if random_color:
```

```
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

fig, axes = plt.subplots()

axes.imshow(np.array(image))
show_mask(medsam_seg, axes)
axes.title.set_text(f"Predicted mask")
axes.axis("off")
```

Out[44]:

```
(-0.5, 255.5, 255.5, -0.5)
```

Predicted mask



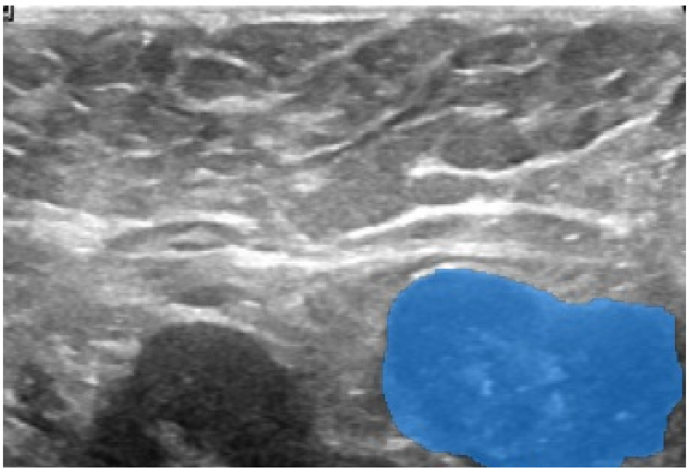

In [45]:

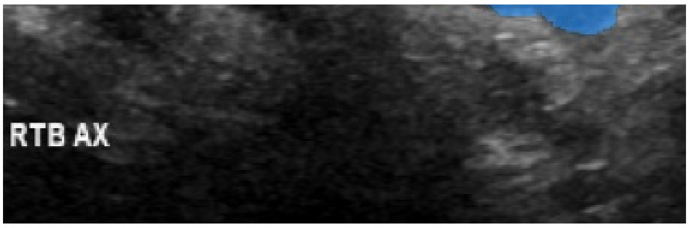
```
fig, axes = plt.subplots()

axes.imshow(np.array(image))
show_mask(ground_truth_mask, axes)
axes.title.set_text(f"Ground truth mask")
axes.axis("off")
```

Out[45]:

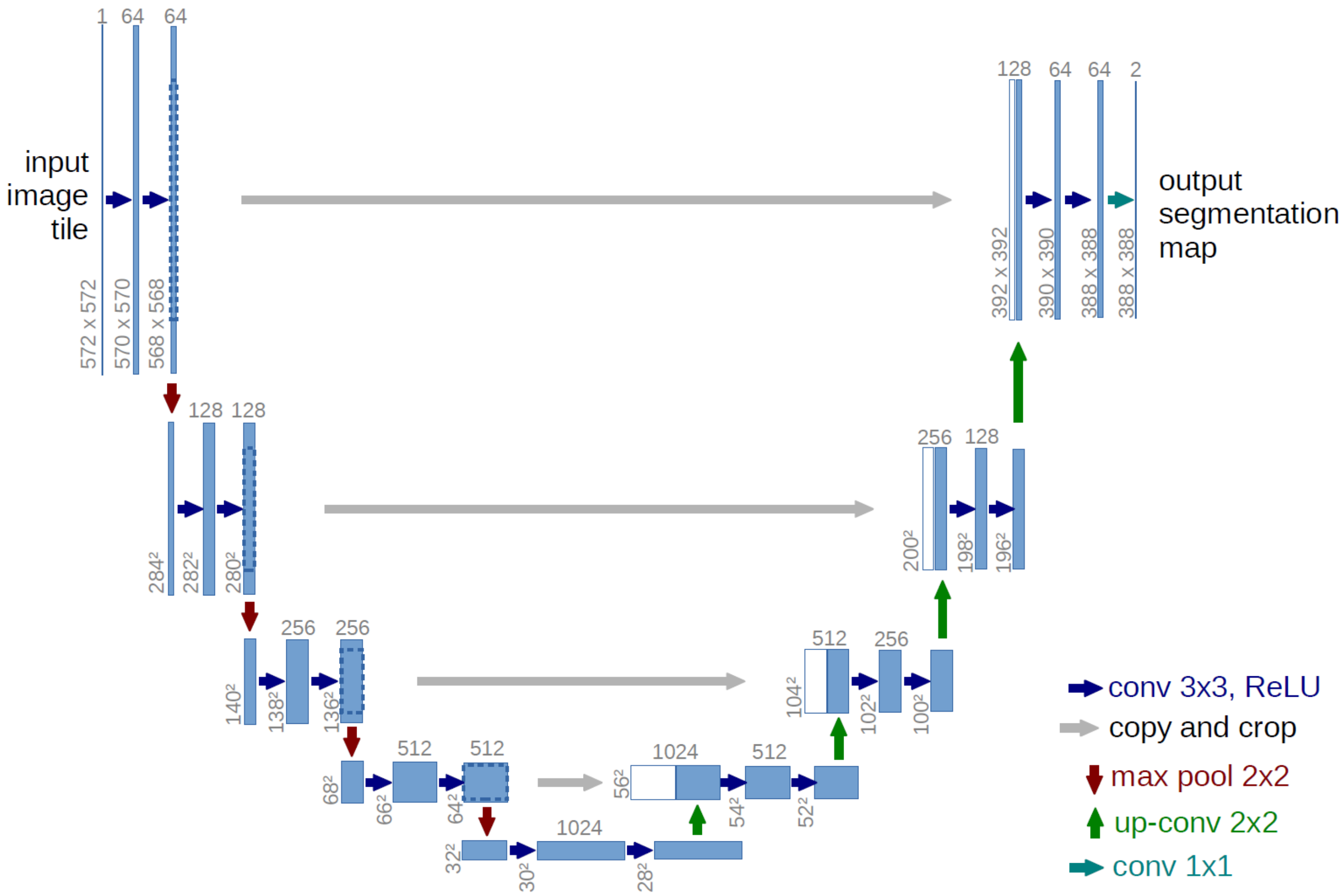
```
(-0.5, 255.5, 255.5, -0.5)
```

Ground truth mask






## UNET modelling




In [47]:

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.conv(x)
```

In [48]:

```
class UNET(nn.Module):
    def __init__(
            self, in_channels=3, out_channels=1, features=[64, 128, 256, 512],
    ):
        super(UNET, self).__init__()
        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
```

```
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Down part of UNET
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET
        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            self.ups.append(DoubleConv(feature*2, feature))

        self.bottleneck = DoubleConv(features[-1], features[-1]*2)
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)

    def forward(self, x):
            skip_connections = []

            for down in self.downs:
                x = down(x)
                skip_connections.append(x)
                x = self.pool(x)

            x = self.bottleneck(x)
            skip_connections = skip_connections[::-1]

            for idx in range(0, len(self.ups), 2):
                x = self.ups[idx](x)
                skip_connection = skip_connections[idx//2]

                if x.shape != skip_connection.shape:
                    x = TF.resize(x, size=skip_connection.shape[2:])

                concat_skip = torch.cat((skip_connection, x), dim=1)
                x = self.ups[idx+1](concat_skip)

            return self.final_conv(x)
```

In [49]:

```
def test():
    x = torch.randn((3, 1, 161, 161))
    model = UNET(in_channels=1, out_channels=1)
    preds = model(x)
    assert preds.shape == x.shape
```

In [50]:

```
class UNET_Dataset(Dataset):
    def __init__(self, image_dir, mask_dir, transform=None):
        self.image_dir = image_dir
        self.mask_dir = mask_dir
        self.transform = transform
        self.images = os.listdir(image_dir)

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

    def __getitem__(self, index):
        img_path = os.path.join(self.image_dir, self.images[index])
        mask_path = os.path.join(self.mask_dir,self.images[index] )
        if img_path and mask_path:
          image = np.array(Image.open(img_path).convert("RGB"))
          mask = np.array(Image.open(mask_path).convert("L"), dtype=np.float32)
          mask[mask == 255.0] = 1.0

        if self.transform is not None:
```

```
            augmentations = self.transform(image=image, mask=mask)
            image = augmentations["image"]
            mask = augmentations["mask"]

        return image, mask
```

In [92]:

```
def save_checkpoint(state, filename="my_checkpoint.pth.tar"):
    print("=> Saving checkpoint")
    torch.save(state, filename)

def load_checkpoint(checkpoint, model):
    print("=> Loading checkpoint")
    model.load_state_dict(checkpoint["state_dict"])

def get_loaders(
    train_dir,
    train_maskdir,
    val_dir,
    val_maskdir,
    batch_size,
    train_transform,
    val_transform,
    num_workers=4,
    pin_memory=True,
):
    train_ds = UNET_Dataset(
        image_dir=train_dir,
        mask_dir=train_maskdir,
        transform=train_transform,
    )

    train_loader = DataLoader(
        train_ds,
        batch_size=batch_size,
        num_workers=num_workers,
        pin_memory=pin_memory,
        shuffle=True,
    )

    val_ds = UNET_Dataset(
        image_dir=val_dir,
        mask_dir=val_maskdir,
        transform=val_transform,
    )

    val_loader = DataLoader(
        val_ds,
        batch_size=batch_size,
        num_workers=num_workers,
        pin_memory=pin_memory,
        shuffle=False,
    )


    return train_loader, val_loader

def check_accuracy(loader, model, lst, device="cuda"):
    num_correct = 0
    num_pixels = 0
    dice_score = 0
    model.eval()

    with torch.no_grad():
        for x, y in loader:
            x = x.to(device)
            y = y.to(device).unsqueeze(1)
            preds = torch.sigmoid(model(x))
            preds = (preds > 0.5).float()
            num_correct += (preds == y).sum()
```

```
            num_pixels += torch.numel(preds)
            dice_score += (2 * (preds * y).sum()) / (
                (preds + y).sum() + 1e-8
            )

    print(
        f"Got {num_correct}/{num_pixels} with acc {num_correct/num_pixels*100:.2f}"
    )
    print(f"Dice score: {dice_score/len(loader)}")
    lst.append([dice_score/len(loader),num_correct/num_pixels*100])
    model.train()

def save_predictions_as_imgs(
    loader, model, folder="saved_images/", device="cuda"
):
    model.eval()
    for idx, (x, y) in enumerate(loader):
        x = x.to(device=device)
        with torch.no_grad():
            preds = torch.sigmoid(model(x))
            preds = (preds > 0.5).float()
        torchvision.utils.save_image(
            preds, f"{folder}/pred_{idx}.png"
        )
        torchvision.utils.save_image(y.unsqueeze(1), f"{folder}{idx}.png")

    model.train()
```

In [52]:

```
!mkdir saved_images
```

In [103]:

```
# Hyperparameters etc.
LEARNING_RATE = 1e-4
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
BATCH_SIZE = 20
NUM_EPOCHS = 50
NUM_WORKERS = 0
IMAGE_HEIGHT = 160  # 1280 originally
IMAGE_WIDTH = 240  # 1918 originally
PIN_MEMORY = True
LOAD_MODEL = False
TRAIN_IMG_DIR = "img"
TRAIN_MASK_DIR = "mask"
VAL_IMG_DIR = "img"
VAL_MASK_DIR = "mask"

def train_fn(loader, model, optimizer, loss_fn, scaler):
    loop = tqdm(loader)
    losss = 0

    for batch_idx, (data, targets) in enumerate(loop):
        data = data.to(device=DEVICE)
        targets = targets.float().unsqueeze(1).to(device=DEVICE)

        # forward
        with torch.cuda.amp.autocast():
            predictions = model(data)
            loss = loss_fn(predictions, targets)
            losss = loss

        # backward
        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        # update tqdm loop
        loop.set_postfix(loss=loss.item())
```

```
    return losss
```

In [104]:

```
unet_loss = []
train_transform = A.Compose(
        [
            A.Resize(height=IMAGE_HEIGHT, width=IMAGE_WIDTH),
            A.Rotate(limit=35, p=1.0),
            A.HorizontalFlip(p=0.5),
            A.VerticalFlip(p=0.1),
            A.Normalize(
                mean=[0.0, 0.0, 0.0],
                std=[1.0, 1.0, 1.0],
                max_pixel_value=255.0,
            ),
            ToTensorV2(),
        ],)

val_transforms = A.Compose(
        [
            A.Resize(height=IMAGE_HEIGHT, width=IMAGE_WIDTH),
            A.Normalize(
                mean=[0.0, 0.0, 0.0],
                std=[1.0, 1.0, 1.0],
                max_pixel_value=255.0,
            ),
            ToTensorV2(),
        ],
    )

model_UNET = UNET(in_channels=3, out_channels=1).to(DEVICE)
loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model_UNET.parameters(), lr=LEARNING_RATE)

train_loader, val_loader = get_loaders(
        TRAIN_IMG_DIR,
        TRAIN_MASK_DIR,
        VAL_IMG_DIR,
        VAL_MASK_DIR,
        BATCH_SIZE,
        train_transform,
        val_transforms,
        NUM_WORKERS,
        PIN_MEMORY,
    )

data = []

if LOAD_MODEL:
        load_checkpoint(torch.load("my_checkpoint.pth.tar"), model_UNET)

print(train_loader,val_loader)
check_accuracy(val_loader, model_UNET, unet_loss, device=DEVICE)
scaler = torch.cuda.amp.GradScaler()

for epoch in range(NUM_EPOCHS):
        a = train_fn(train_loader, model_UNET, optimizer, loss_fn, scaler)
        data.append(a)

        # save model
        checkpoint = {
            "state_dict": model_UNET.state_dict(),
            "optimizer":optimizer.state_dict(),
        }
        save_checkpoint(checkpoint)

        # check accuracy
        check_accuracy(val_loader, model_UNET,unet_loss, device=DEVICE)
```

```
        # print some examples to a folder
        save_predictions_as_imgs(
            val_loader, model_UNET, folder="saved_images/", device=DEVICE
)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f96a0b7cd30> <torch.utils.data.datal
oader.DataLoader object at 0x7f96a0b7f6d0>
Got 4307618/4992000 with acc 86.29
Dice score: 0.0
```

```
100%|██████████| 7/7 [00:02<00:00,  2.84it/s, loss=0.6]
```

```
=> Saving checkpoint
Got 4307618/4992000 with acc 86.29
Dice score: 0.0
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.531]
```

```
=> Saving checkpoint
Got 4307618/4992000 with acc 86.29
Dice score: 0.0
```

```
100%|██████████| 7/7 [00:02<00:00,  2.67it/s, loss=0.464]
```

```
=> Saving checkpoint
Got 4307618/4992000 with acc 86.29
Dice score: 0.0
```

```
100%|██████████| 7/7 [00:02<00:00,  2.77it/s, loss=0.441]
```

```
=> Saving checkpoint
Got 4307618/4992000 with acc 86.29
Dice score: 0.0
```

```
100%|██████████| 7/7 [00:02<00:00,  2.50it/s, loss=0.371]
```

```
=> Saving checkpoint
Got 4373534/4992000 with acc 87.61
Dice score: 0.2180602252483368
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.459]
```

```
=> Saving checkpoint
Got 4510421/4992000 with acc 90.35
Dice score: 0.5493773221969604
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.44]
```

```
=> Saving checkpoint
Got 4521334/4992000 with acc 90.57
Dice score: 0.604794979095459
```

```
100%|██████████| 7/7 [00:02<00:00,  2.75it/s, loss=0.384]
```

```
=> Saving checkpoint
Got 4449588/4992000 with acc 89.13
Dice score: 0.558542788028717
```

```
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.329]
```

```
=> Saving checkpoint
Got 4551908/4992000 with acc 91.18
Dice score: 0.6693931818008423
```

```
100%|██████████| 7/7 [00:02<00:00,  2.52it/s, loss=0.35]
```

```
=> Saving checkpoint
Got 4464802/4992000 with acc 89.44
Dice score: 0.6363086700439453
```

```
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.32]
```

```
=> Saving checkpoint
Got 4535576/4992000 with acc 90.86
Dice score: 0.6630691885948181
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.363]
```

```
=> Saving checkpoint
Got 4560862/4992000 with acc 91.36
Dice score: 0.5571573376655579
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.311]
=> Saving checkpoint
Got 4576707/4992000 with acc 91.68
Dice score: 0.585880696773529
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.335]
=> Saving checkpoint
Got 4605274/4992000 with acc 92.25
Dice score: 0.6282346844673157
100%|██████████| 7/7 [00:02<00:00,  2.51it/s, loss=0.315]
=> Saving checkpoint
Got 4603880/4992000 with acc 92.23
Dice score: 0.6866751313209534
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.373]
=> Saving checkpoint
Got 4606184/4992000 with acc 92.27
Dice score: 0.7114956974983215
100%|██████████| 7/7 [00:02<00:00,  2.69it/s, loss=0.328]
=> Saving checkpoint
Got 4555815/4992000 with acc 91.26
Dice score: 0.6692947149276733
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.314]
=> Saving checkpoint
Got 4624328/4992000 with acc 92.63
Dice score: 0.7022212743759155
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.271]
=> Saving checkpoint
Got 4637818/4992000 with acc 92.91
Dice score: 0.6904920339584351
100%|██████████| 7/7 [00:02<00:00,  2.61it/s, loss=0.351]
=> Saving checkpoint
Got 4651762/4992000 with acc 93.18
Dice score: 0.7252456545829773
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.29]
=> Saving checkpoint
Got 4617021/4992000 with acc 92.49
Dice score: 0.6457382440567017
100%|██████████| 7/7 [00:02<00:00,  2.60it/s, loss=0.313]
=> Saving checkpoint
Got 4585578/4992000 with acc 91.86
Dice score: 0.5727750062942505
100%|██████████| 7/7 [00:02<00:00,  2.84it/s, loss=0.3]
=> Saving checkpoint
Got 4641185/4992000 with acc 92.97
Dice score: 0.6780946850776672
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.282]
=> Saving checkpoint
Got 4643256/4992000 with acc 93.01
Dice score: 0.680303156375885
```

```
100%|██████████| 7/7 [00:02<00:00,  2.64it/s, loss=0.282]
```

```
=> Saving checkpoint
Got 4648877/4992000 with acc 93.13
Dice score: 0.7286368608474731
```

```
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.297]
```

```
=> Saving checkpoint
Got 4692315/4992000 with acc 94.00
Dice score: 0.7388197779655457
```

```
100%|██████████| 7/7 [00:02<00:00,  2.54it/s, loss=0.311]
```

```
=> Saving checkpoint
Got 4707853/4992000 with acc 94.31
Dice score: 0.7479079365730286
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.247]
```

```
=> Saving checkpoint
Got 4589345/4992000 with acc 91.93
Dice score: 0.6246888637542725
```

```
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.246]
```

```
=> Saving checkpoint
Got 4699480/4992000 with acc 94.14
Dice score: 0.7438232898712158
```

```
100%|██████████| 7/7 [00:02<00:00,  2.68it/s, loss=0.267]
```

```
=> Saving checkpoint
Got 4709830/4992000 with acc 94.35
Dice score: 0.7651509046554565
```

```
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.273]
```

```
=> Saving checkpoint
Got 4673356/4992000 with acc 93.62
Dice score: 0.7274051904678345
```

```
100%|██████████| 7/7 [00:02<00:00,  2.54it/s, loss=0.281]
```

```
=> Saving checkpoint
Got 4682761/4992000 with acc 93.81
Dice score: 0.71808260679245
```

```
100%|██████████| 7/7 [00:02<00:00,  2.81it/s, loss=0.246]
```

```
=> Saving checkpoint
Got 4682762/4992000 with acc 93.81
Dice score: 0.7282747030258179
```

```
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.294]
```

```
=> Saving checkpoint
Got 4672720/4992000 with acc 93.60
Dice score: 0.7156001925468445
```

```
100%|██████████| 7/7 [00:02<00:00,  2.74it/s, loss=0.261]
```

```
=> Saving checkpoint
Got 4688685/4992000 with acc 93.92
Dice score: 0.7389267683029175
```

```
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.287]
```

```
=> Saving checkpoint
Got 4691836/4992000 with acc 93.99
Dice score: 0.744437038898468
```

```
100%|██████████| 7/7 [00:02<00:00,  2.51it/s, loss=0.342]
```

```
=> Saving checkpoint
Got 4689295/4992000 with acc 93.94
Dice score: 0.7644223570823669
```

```
100%|██████████| 7/7 [00:02<00:00,  2.80it/s, loss=0.313]
=> Saving checkpoint
Got 4630293/4992000 with acc 92.75
Dice score: 0.7240407466888428
100%|██████████| 7/7 [00:02<00:00,  2.76it/s, loss=0.353]
=> Saving checkpoint
Got 4543496/4992000 with acc 91.02
Dice score: 0.7188430428504944
100%|██████████| 7/7 [00:02<00:00,  2.79it/s, loss=0.301]
=> Saving checkpoint
Got 4595156/4992000 with acc 92.05
Dice score: 0.6678043603897095
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.293]
=> Saving checkpoint
Got 4675965/4992000 with acc 93.67
Dice score: 0.7460671067237854
100%|██████████| 7/7 [00:02<00:00,  2.52it/s, loss=0.236]
=> Saving checkpoint
Got 4693329/4992000 with acc 94.02
Dice score: 0.7348936200141907
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.246]
=> Saving checkpoint
Got 4697053/4992000 with acc 94.09
Dice score: 0.7307611107826233
100%|██████████| 7/7 [00:02<00:00,  2.72it/s, loss=0.262]
=> Saving checkpoint
Got 4700224/4992000 with acc 94.16
Dice score: 0.7378231883049011
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.222]
=> Saving checkpoint
Got 4688433/4992000 with acc 93.92
Dice score: 0.7329518795013428
100%|██████████| 7/7 [00:02<00:00,  2.82it/s, loss=0.22]
=> Saving checkpoint
Got 4718803/4992000 with acc 94.53
Dice score: 0.7581741213798523
100%|██████████| 7/7 [00:02<00:00,  2.55it/s, loss=0.236]
=> Saving checkpoint
Got 4695544/4992000 with acc 94.06
Dice score: 0.7417799234390259
100%|██████████| 7/7 [00:02<00:00,  2.78it/s, loss=0.226]
=> Saving checkpoint
Got 4704884/4992000 with acc 94.25
Dice score: 0.7891114950180054
100%|██████████| 7/7 [00:02<00:00,  2.67it/s, loss=0.25]
=> Saving checkpoint
Got 4755193/4992000 with acc 95.26
Dice score: 0.806406557559967
100%|██████████| 7/7 [00:02<00:00,  2.83it/s, loss=0.221]
=> Saving checkpoint
Got 4721705/4992000 with acc 94.59
```

Dice score: 0.759075403213501

In [106]:

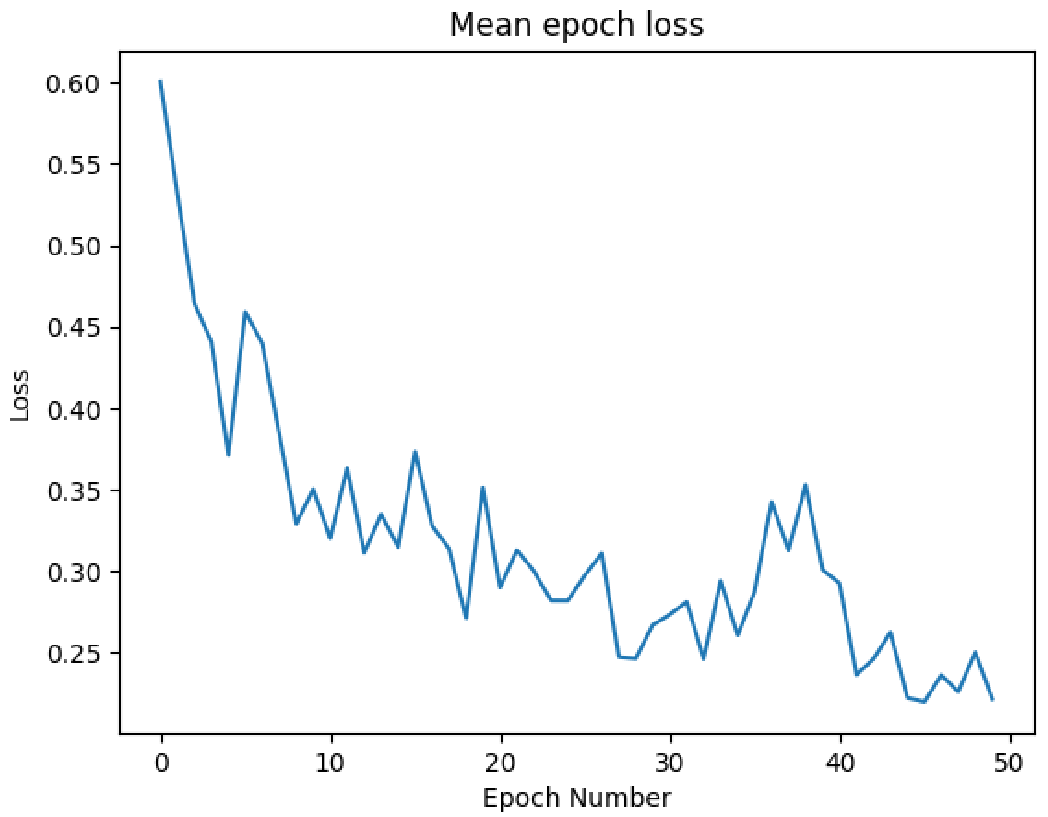
```
data[0].item()
```

Out[106]:

0.6004032492637634

In [107]:

```
mean_losses = [x.item() for x in data]

plt.plot(list(range(len(mean_losses))), mean_losses)
plt.title('Mean epoch loss')
plt.xlabel('Epoch Number')
plt.ylabel('Loss')

plt.show()
```

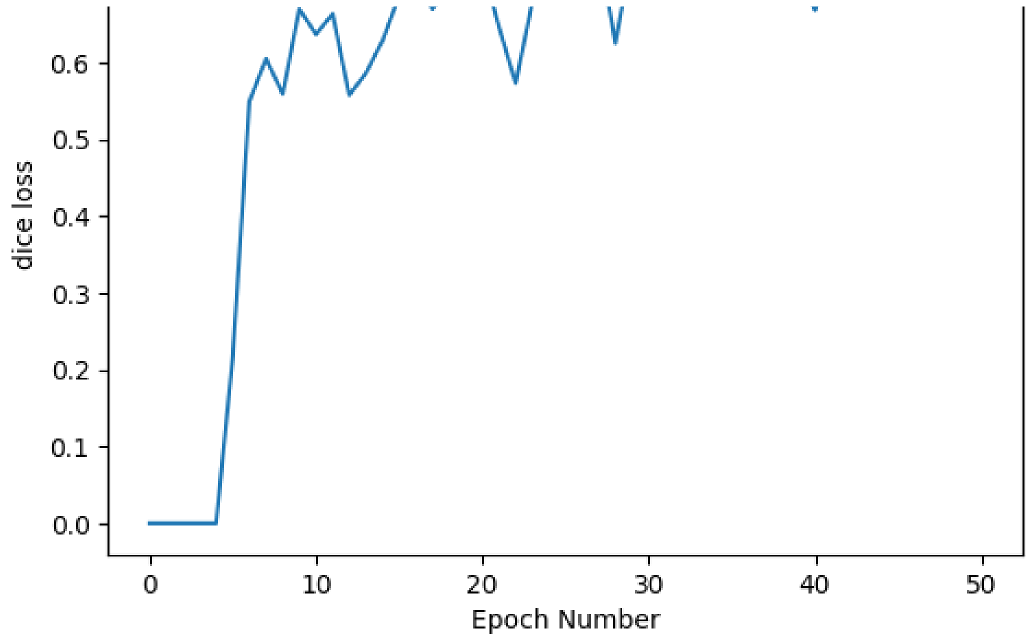



In [121]:

```
dice = []
acc = []
for x in unet_loss:
  dice.append(x[0].item())
  acc.append(x[1].item())
```

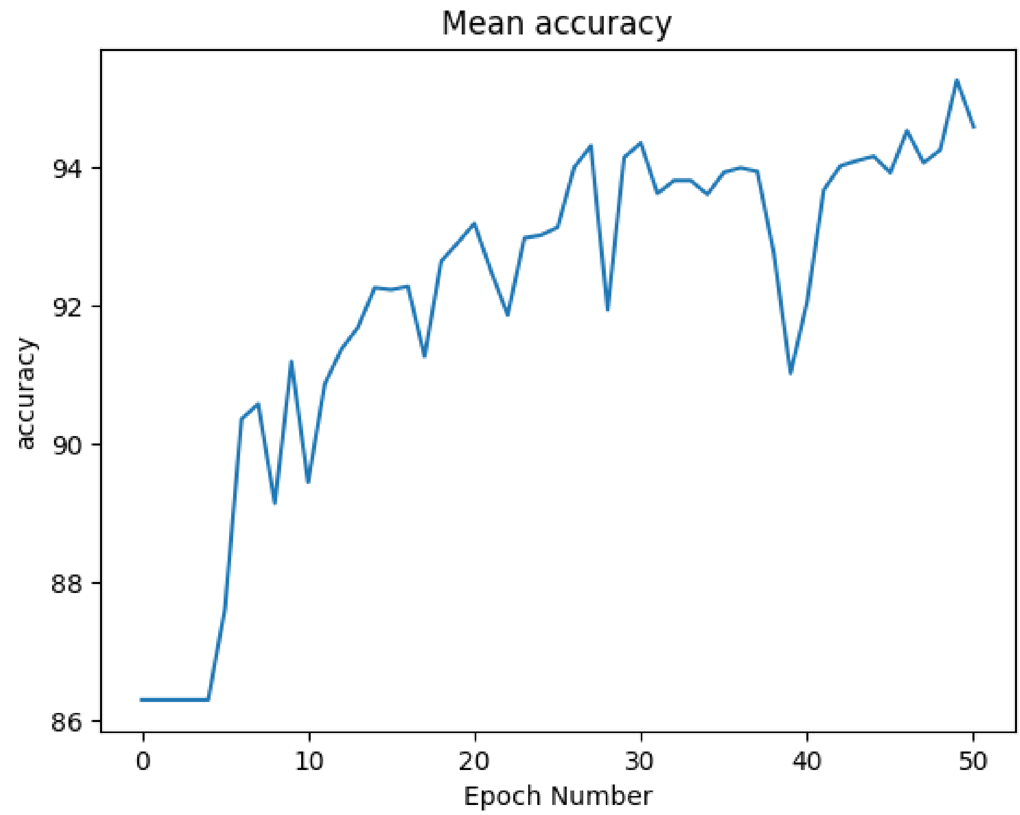
In [122]:

```
plt.plot(list(range(len(dice))), dice)
plt.title('Mean dice loss')
plt.xlabel('Epoch Number')
plt.ylabel('dice loss')

plt.show()
```




In [123]:

```
plt.plot(list(range(len(acc))), acc)
plt.title('Mean accuracy')
plt.xlabel('Epoch Number')
plt.ylabel('accuracy')

plt.show()
```




In [124]:

```
def plot_images(images, mask = False):
    fig = plt.figure(figsize=(10, 8))
    columns = len(images)
    rows = 1
    for i in range(columns*rows):
        img = images[i]
        fig.add_subplot(rows, columns, i+1)
        if mask:
            img = img > 0.5
        plt.imshow(img, cmap="gray")
```

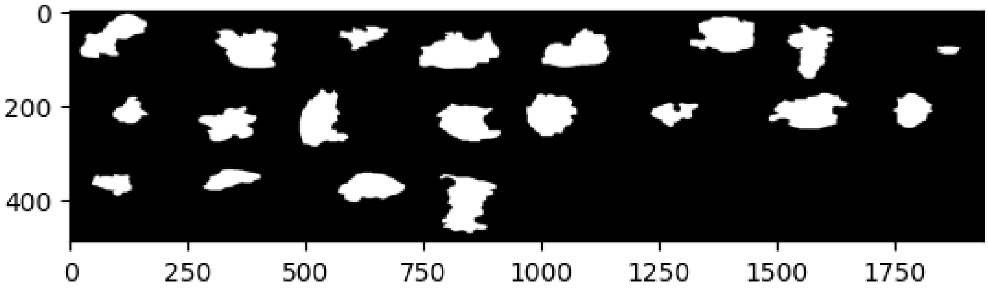
```
    plt.show()
```

In [145]:

```
ground_t = cv2.imread('saved_images/0.png')
plt.imshow(np.array(groud_t))
```

Out[145]:

<matplotlib.image.AxesImage at 0x7f96ac170b20>

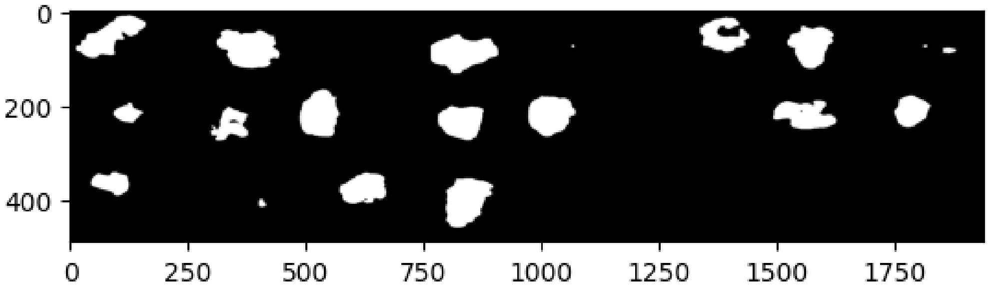



In [146]:

```
pred_t = cv2.imread('saved_images/pred_0.png')
plt.imshow(np.array(pred_t))
```

Out[146]:

<matplotlib.image.AxesImage at 0x7f96a089d1e0>




In [156]:

```
for x, y in val_loader:
        x = x.to(device)
```

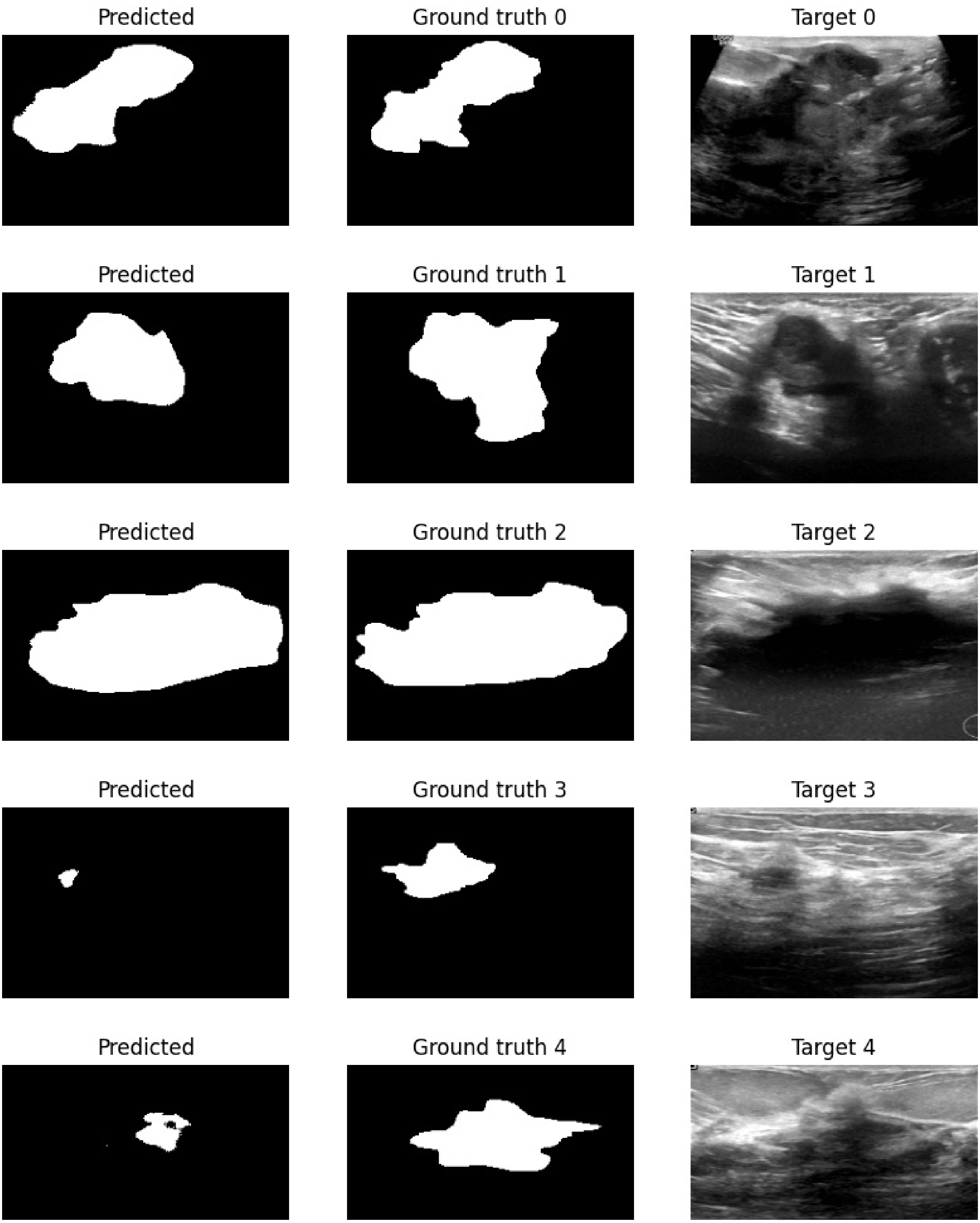
In [171]:

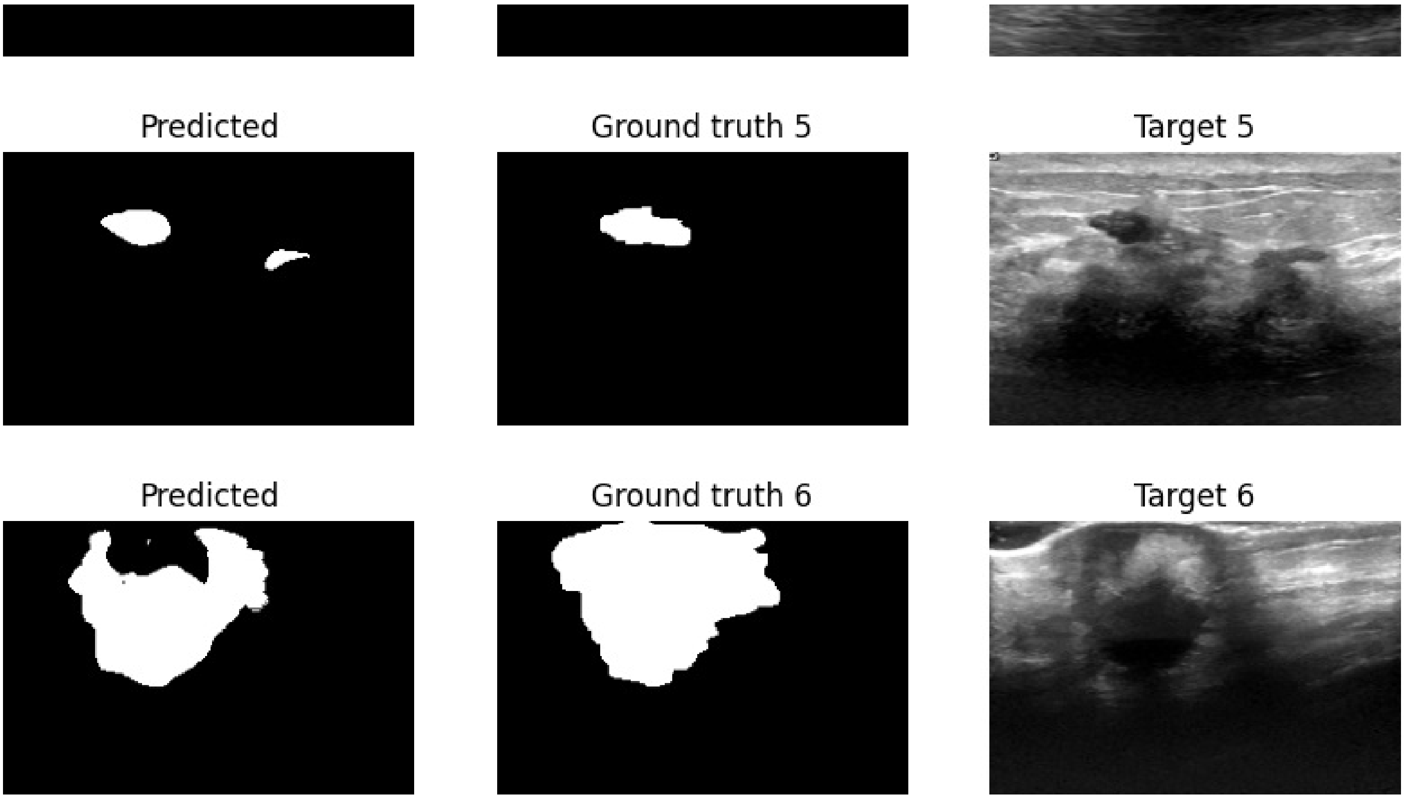
```
def show_img(loader, model, device="cuda"):
    model.eval()
    for idx, (x, y) in enumerate(loader):
        x = x.to(device=device)
        with torch.no_grad():
            preds = torch.sigmoid(model(x))
            preds = (preds > 0.5).float()

        # Convert tensors to numpy arrays
        y = y.unsqueeze(1)
        preds_np = preds.detach().cpu().numpy()
        y_np = y.detach().cpu().numpy()
        x_np = x.detach().cpu().numpy()


        # Display the predicted and target images
        plt.figure(figsize=(10, 5))
        plt.subplot(1, 3, 1)
        plt.imshow(preds_np[0, 0], cmap='gray')
        plt.title("Predicted")
        plt.axis('off')

        plt.subplot(1, 3, 2)
```

```
        plt.imshow(y_np[0, 0], cmap='gray')
        plt.title(f"Ground truth {idx}")
        plt.axis('off')

        plt.subplot(1, 3, 3)
        plt.imshow(x_np[0, 0], cmap='gray')
        plt.title(f"Target {idx}")
        plt.axis('off')

        plt.show()
```

In [172]:

```
show_img(val_loader,model_UNET)
```




Predicted
Ground truth 5
Target 5
Predicted
Ground truth 6
Target 6