Project work: A mini segmentation challenge

Imaging for the Life Sciences
MSLS / CO4: Project work

Student: ⇒ Guansheng Du
University: ⇒ ZHAW
Semester: ⇒ SS24
Date: ⇒ 26.05.2024

Table of contents

Introduction

Lower-grade gliomas (LGG), classified as WHO grade II and III brain tumors, are the focus of this mini project. I developed a neural network-based program to segment the cancerous regions from a series of MRI brain images. This dataset comprises brain MR images along with manually segmented FLAIR abnormality masks.

The images are associated with 110 patients from The Cancer Genome Atlas (TCGA) lower-grade glioma collection, all of whom have at least one fluid-attenuated inversion recovery (FLAIR) sequence. Out of these, 101 patients had all sequences available (pre-contrast, FLAIR, and post-contrast sequences), 9 patients lacked the post-contrast sequence, and 6 were missing the pre-contrast sequence.

In this project, images known to have valid tumor masks were loaded into a dataframe `df_mask`. A total of 1373 images were loaded into the dataset, with 80% used for training and the remaining 20% used for validation. These images were augmented and then trained using a U-Net architecture deep learning model. The Adam optimizer was used for optimization, and the Dice coefficient was used as the loss function. The learning rate was set to 0.001. After 50 training epochs, the model achieved a loss of 0.1628 on the training dataset and 0.1807 on the validation dataset. This small course project demonstrates the potential for segmenting tumor regions from brain MRI images.

Prerequisites / Setup

⇒ Special setup instructions, imports and configurations go here.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import nibabel as nib
import pydicom
import PIL
from PIL import Image
import os
import pandas as pd

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.transforms.v2 as transforms
from torch import optim

from torch.utils.data import Dataset


# Jupyter / IPython configuration:
# Automatically reload modules when modified
%load_ext autoreload
```

```
%autoreload 2

# Enable vectorized output (for nicer plots)
%config InlineBackend.figure_formats = ["svg"]

# Inline backend configuration
%matplotlib inline

# Enable this line if you want to use the interactive widgets
# It requires the ipympl package to be installed.
#%matplotlib widget

data_folder = '/home/gs/Desktop/SS24 CO4 Imaging for the Life Sciences/MRI_segmentation_data/raw/'
```

Help functions

In [2]:
```
# Sort filenames in natural order
# Generated by AI

import re

def natural_sort(l):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split("([0-9]+)", key)]
    return sorted(l, key=alphanum_key)
```

In [3]:
```
# Check if the file has a mask

def check_mask(file):
    # Open the file
    image = Image.open(file)

    # Convert the image to an array
    image_array = np.array(image)

    # Check if there is a white pixel
    if np.any(image_array > 0):
        return 1
```

```
    else:
        return 0
```

In [4]:
```
def overlay_mask(image, mask):
    image = np.array(image)
    mask = np.array(mask)
    mask = mask > 0.5
    mask = mask.astype(np.uint8)

    mask = mask * 255

    # Find contours in the mask
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    cv2.drawContours(image, contours, -1, (0, 255, 0), 1)

    return image
```

In [5]:
```
from skimage.transform import resize

def overlay_output(image, output):
    image = resize(image, output.shape)
    image = np.array(image)
    output = np.array(output)
    output = output > 0.5
    output = output.astype(np.uint8)

    output = output * 255

    # Find contours in the mask
    contours, _ = cv2.findContours(output, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    cv2.drawContours(image, contours, -1, (255, 0, 0), 1)

    return image
```

In [6]:
```
# Display the 16 images with their masks

def image_plots(df, indices):
    fig, axs = plt.subplots(4, 4, figsize=(10, 10))
```

```
    for i, index in enumerate(indices):
        image = Image.open(df.iloc[index]["image_path"])
        mask = Image.open(df.iloc[index]["mask_path"])

        ax = axs[i // 4, i % 4]
        overlay = overlay_mask(image, mask)
        ax.imshow(overlay)
        # ax.imshow(image)
        # ax.imshow(mask, cmap="jet", alpha=0.5)
        ax.axis("off")

        mask_status = "Mask" if df.iloc[index]["check_mask"] == 1 else "No mask"
        ax.set_title(f"Index: {index}\n{mask_status}")

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

In [7]:
```
def plot_side_by_side(image, mask, pred, title1="Image", title2="Mask", title3="Prediction"):
    image = np.array(image)
    mask = np.array(mask)
    plt.figure(figsize=(8, 5))
    plt.subplot(1, 3, 1)

    plt.imshow(image)
    plt.title(title1)
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(mask, cmap="gray")
    plt.title(title2)
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(pred, cmap="gray")
    plt.title(title3)
    plt.axis("off")

    plt.show()
```

In [8]:
```
def plot_losses(train_loss, val_loss):
    plt.figure(figsize=(8, 5))
    plt.plot(train_loss, label="Training Loss")
    plt.plot(val_loss, label="Testing Loss")
    plt.title("Loss vs Epoch")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()
```

Dataset

Build a DataFrame to store image and its mask path.

In [9]:
```
# Get a list of all .tif files in the data folder
tif_files = [f for f in os.listdir(data_folder) if f.endswith(".tif")]

# Sort the list of files in natural order
tif_files = natural_sort(tif_files)
```

In [10]:
```
# Create a DataFrame

mask_files = [f for f in tif_files if "_mask" in f]
image_files = [f for f in tif_files if "_mask" not in f]

df = pd.DataFrame({"image_file": image_files, "mask_file": mask_files})
df.head()
```

Out[10]:

	image_file	mask_file
0	TCGA_CS_4941_19960909_1.tif	TCGA_CS_4941_19960909_1_mask.tif
1	TCGA_CS_4941_19960909_2.tif	TCGA_CS_4941_19960909_2_mask.tif
2	TCGA_CS_4941_19960909_3.tif	TCGA_CS_4941_19960909_3_mask.tif
3	TCGA_CS_4941_19960909_4.tif	TCGA_CS_4941_19960909_4_mask.tif
4	TCGA_CS_4941_19960909_5.tif	TCGA_CS_4941_19960909_5_mask.tif

In [11]:
```
df['mask_path'] = df.mask_file.apply(lambda x: data_folder + x)
df['image_path'] = df.image_file.apply(lambda x: data_folder + x)

df.head()
```

Out[11]:

	image_file	mask_file	mask_path	image_path
0	TCGA_CS_4941_19960909_1.tif	TCGA_CS_4941_19960909_1_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...
1	TCGA_CS_4941_19960909_2.tif	TCGA_CS_4941_19960909_2_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...
2	TCGA_CS_4941_19960909_3.tif	TCGA_CS_4941_19960909_3_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...
3	TCGA_CS_4941_19960909_4.tif	TCGA_CS_4941_19960909_4_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...
4	TCGA_CS_4941_19960909_5.tif	TCGA_CS_4941_19960909_5_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...

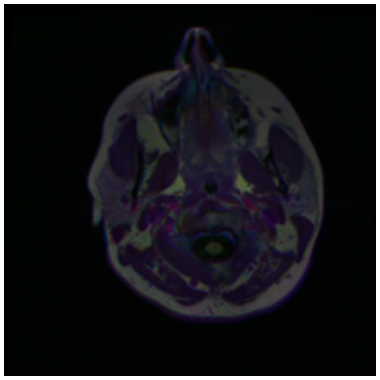
In [12]:
```
df["check_mask"] = df["mask_path"].apply(check_mask)

df.head()
```

Out[12]:

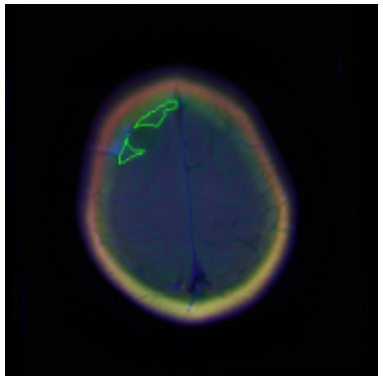
	image_file	mask_file	mask_path	image_path	check_mask
0	TCGA_CS_4941_19960909_1.tif	TCGA_CS_4941_19960909_1_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	0
1	TCGA_CS_4941_19960909_2.tif	TCGA_CS_4941_19960909_2_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	0
2	TCGA_CS_4941_19960909_3.tif	TCGA_CS_4941_19960909_3_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	0
3	TCGA_CS_4941_19960909_4.tif	TCGA_CS_4941_19960909_4_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	0
4	TCGA_CS_4941_19960909_5.tif	TCGA_CS_4941_19960909_5_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	0

In [13]:
```
# Random pick 16 images from the DataFrame and make image plots

# np.random.seed(42)

# Generate 16 random indices
indices = np.random.randint(0, len(df), 16)

image_plots(df, indices)
```

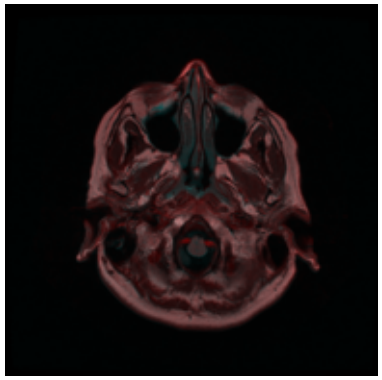
Index: 1571
No mask



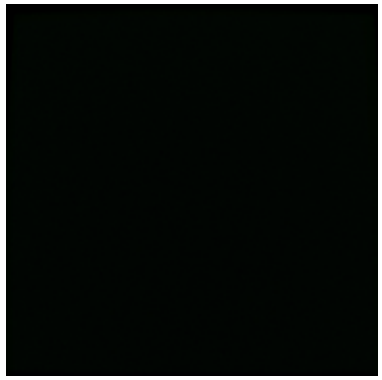
Index: 143
Mask



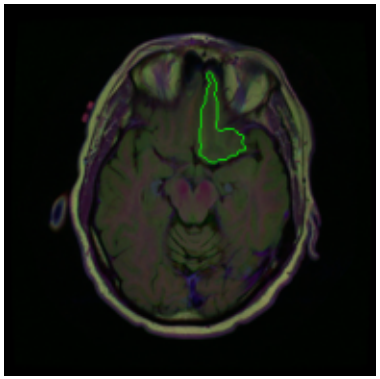
Index: 1841
No mask



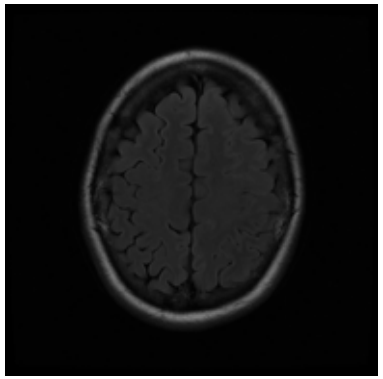
Index: 2088
No mask



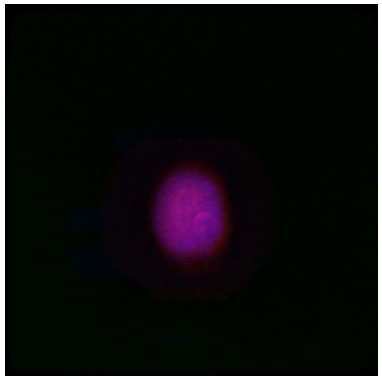
Index: 1619
Mask



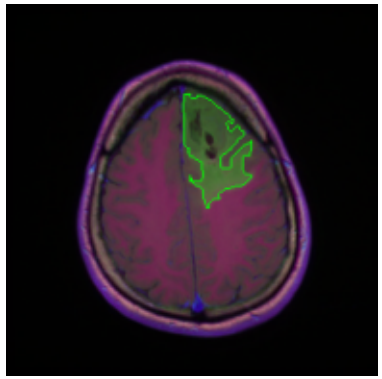
Index: 3923
No mask



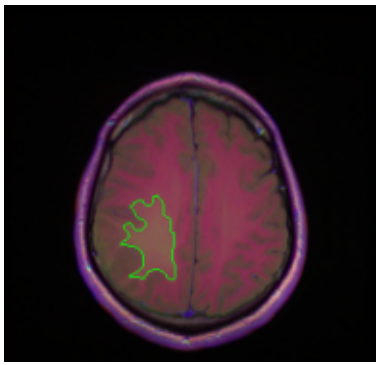
Index: 1718
No mask



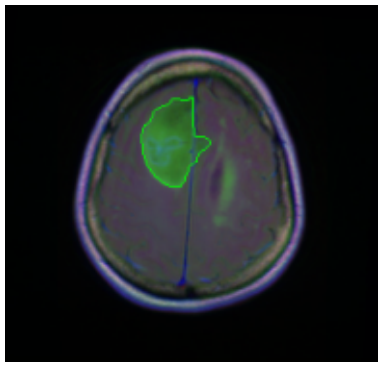
Index: 3270
Mask



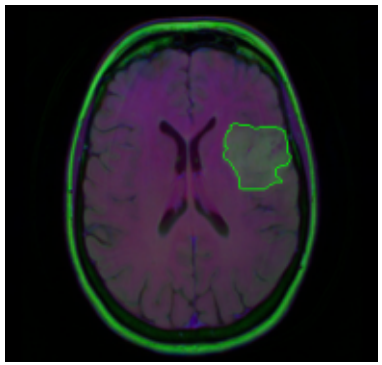
Index: 2922
Mask



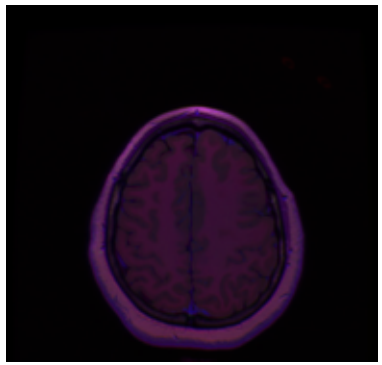
Index: 3199
Mask

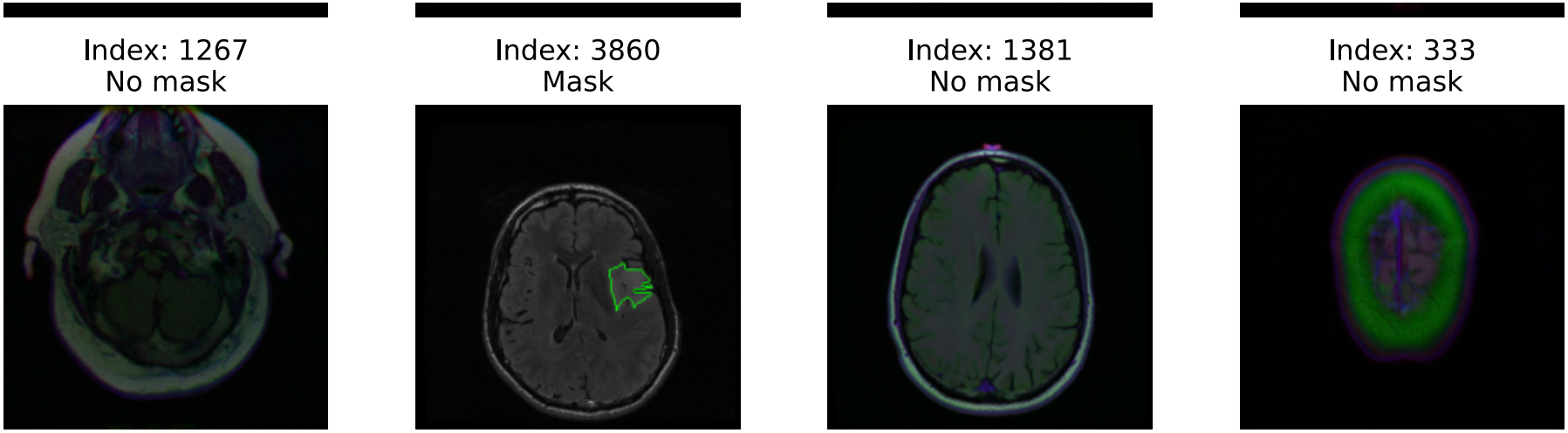


Index: 324
Mask



Index: 3784
No mask





Preprocessing

Augmentations was applied to images before the deep learning training.

In [14]:
```
train_data_transform = transforms.Compose(
    [
        transforms.Resize((128, 128)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomApply([transforms.RandomRotation(degrees=90)], p=0.5),
        transforms.ToImage(),
        transforms.ToDtype(torch.float32, scale=True),
    ]
)

test_data_transform = transforms.Compose(
    [
        transforms.Resize((128, 128)),
        transforms.ToImage(),
        transforms.ToDtype(torch.float32, scale=True),
```

```
    ]
)
```

In [15]:
```
def plot_model_results(image, mask, predict):
    fig, axs = plt.subplots(1, 3, figsize=(10, 3.33))

    axs[0].imshow(transforms.ToPILImage()(image))
    axs[0].set_title("Image")
    axs[0].axis("off")

    axs[1].imshow(transforms.ToPILImage()(mask), cmap="gray")
    axs[1].set_title("Mask")
    axs[1].axis("off")

    axs[2].imshow(transforms.ToPILImage()(predict.squeeze(0)), cmap="gray")
    axs[2].set_title("Predicted mask")
    axs[2].axis("off")

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

Manual segmentation

In this project, a researcher who was a medical school graduate with experience in neuroradiology imaging, manually annotated FLAIR images by drawing an outline of the FLAIR abnormality on each slice to form training data for the automatic segmentation algorithm. For more details about the manual segmentation, please refer the reference.

Automated segmentation

A neural network was applied to automatically segment the images in this project.

Dataset preparation

In [16]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

device
```

Out[16]:
```
device(type='cuda', index=0)
```

Only images with masks were applied for training.

Build `df_mask` for training and testing.

In [17]:
```
df_mask = df[df["check_mask"] == 1]
# train_df = df_mask.sample(frac=0.8, random_state=42)

df_mask.head()
```

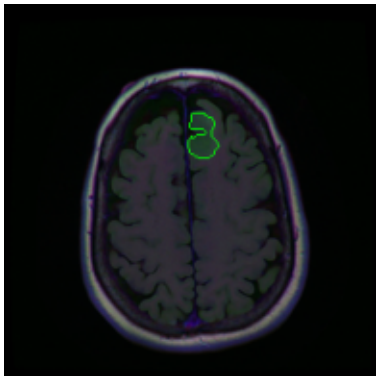
Out[17]:

	image_file	mask_file	mask_path	image_path	check_mask
10	TCGA_CS_4941_19960909_11.tif	TCGA_CS_4941_19960909_11_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	1
11	TCGA_CS_4941_19960909_12.tif	TCGA_CS_4941_19960909_12_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	1
12	TCGA_CS_4941_19960909_13.tif	TCGA_CS_4941_19960909_13_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	1
13	TCGA_CS_4941_19960909_14.tif	TCGA_CS_4941_19960909_14_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	1
14	TCGA_CS_4941_19960909_15.tif	TCGA_CS_4941_19960909_15_mask.tif	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	/home/gs/Desktop/SS24 CO4 Imaging for the Life...	1

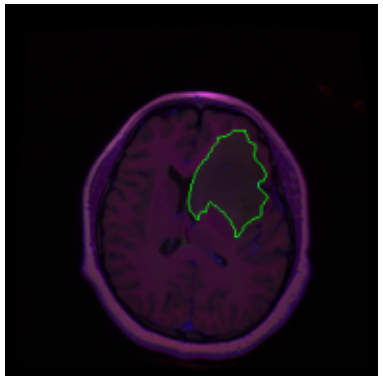
In [18]:
```
# Random pick 16 images from the `df_mask` and make image plots
```

```
# np.random.seed(42)
indices = np.random.randint(0, len(df_mask), 16)
image_plots(df_mask, indices)
```

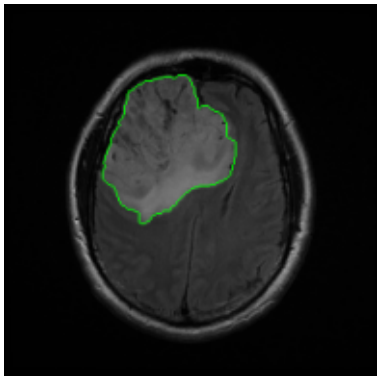
Index: 227
Mask



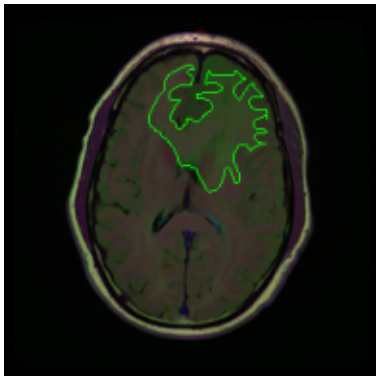
Index: 1323
Mask



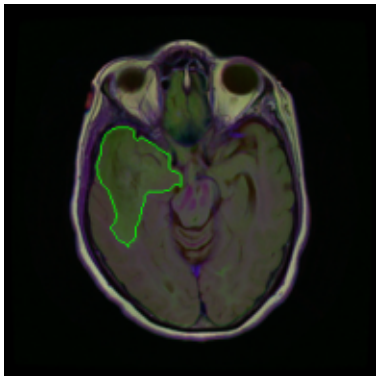
Index: 1221
Mask



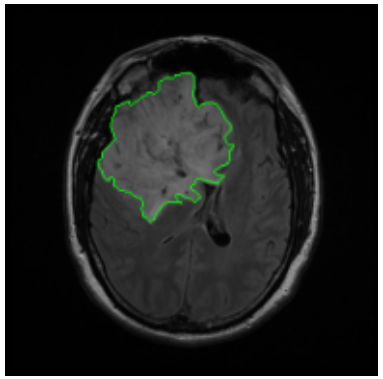
Index: 551
Mask



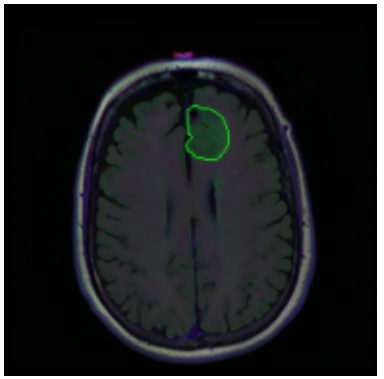
Index: 459
Mask



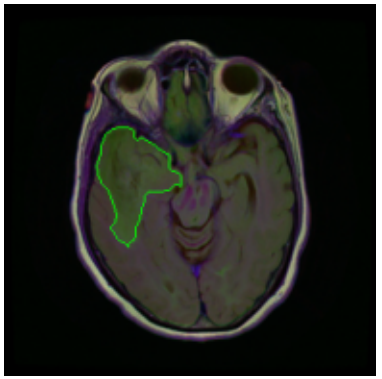
Index: 1217
Mask



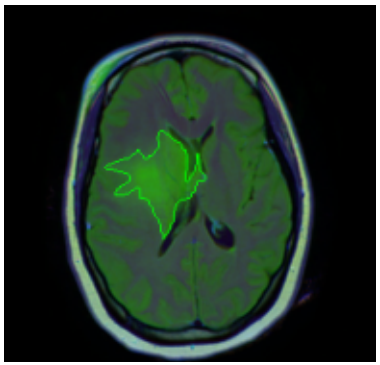
Index: 224
Mask



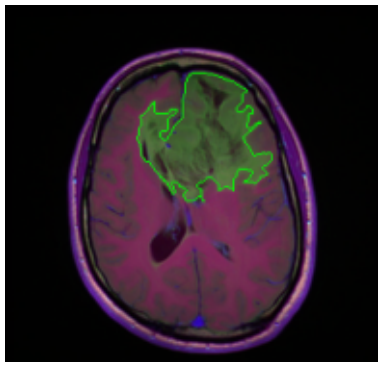
Index: 459
Mask



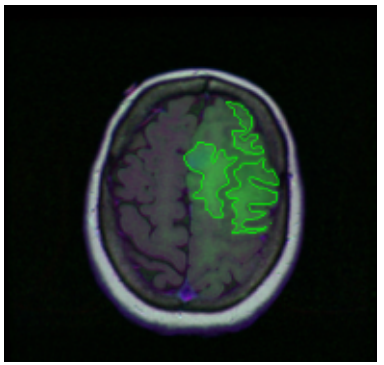
Index: 728
Mask



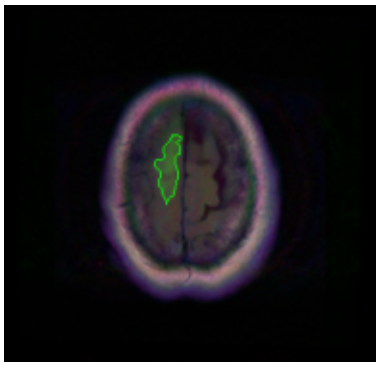
Index: 1120
Mask

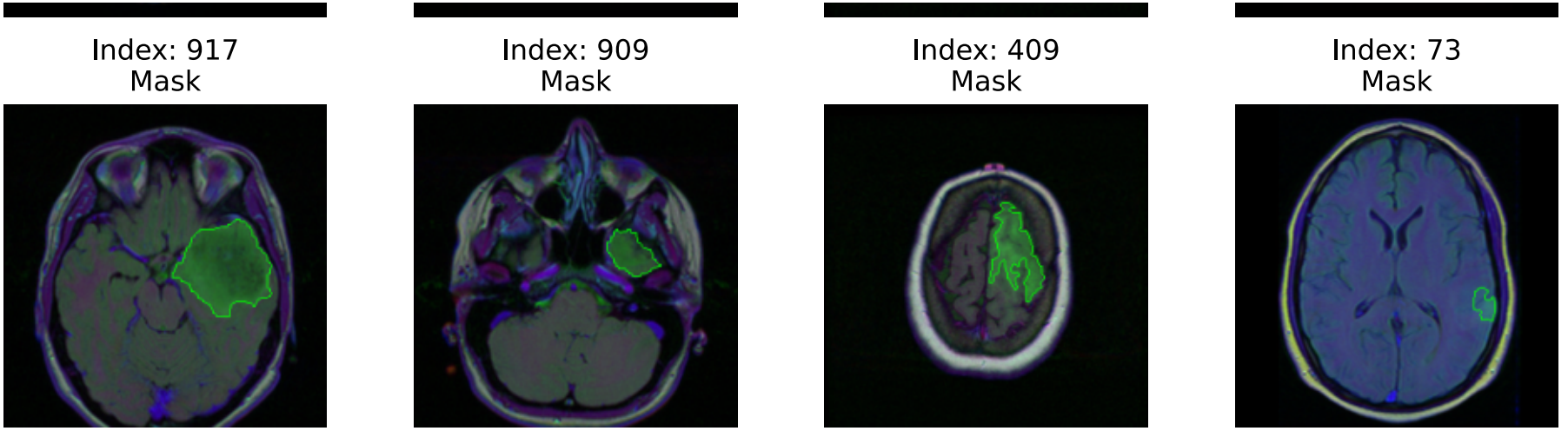


Index: 405
Mask



Index: 1103
Mask



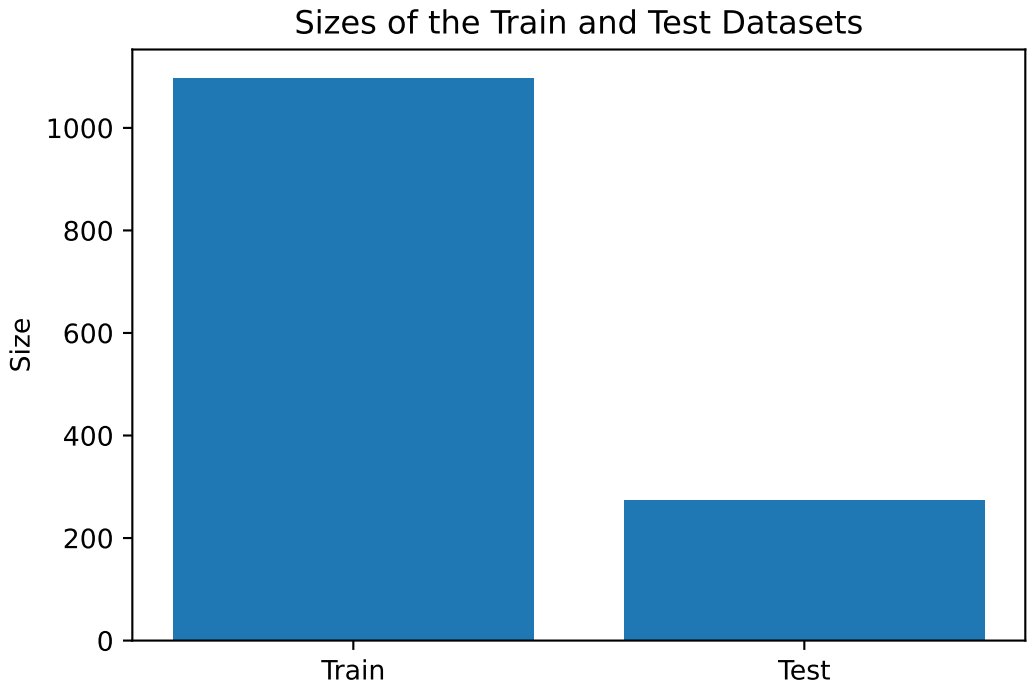


In [19]:
```python
# Class to create a custom dataset

class CustomImageDataset(Dataset):
    def __init__(self, df, transforms=None):
        self.df = df
        self.transforms = transforms

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

    def load_image(self, idx):
        image = Image.open(self.df.iloc[idx]["image_path"])
        mask = Image.open(self.df.iloc[idx]["mask_path"])
        return image, mask

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = self.df.iloc[idx]["image_path"]
        mask_name = self.df.iloc[idx]["mask_path"]

        image = Image.open(img_name)
        mask = Image.open(mask_name)
```

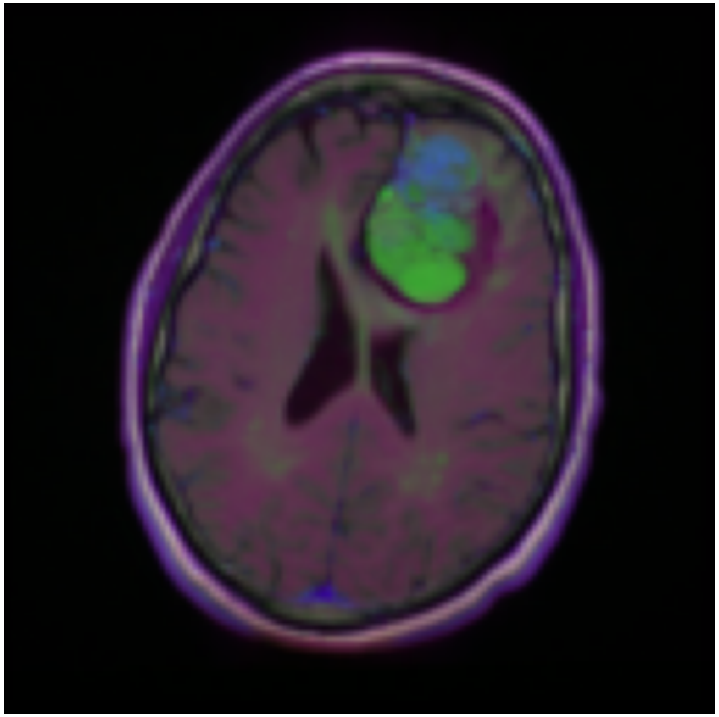
```
        image, mask = self.transforms(image, mask)

        return image, mask
```

In [20]:
```
# np.random.seed(42)

train_df = df_mask.sample(frac=0.8, random_state=np.random.randint(0, 1000000))
test_df = df_mask.drop(train_df.index)

# Save images in the Pytorch dataset format

train_dataset = CustomImageDataset(train_df, transforms=train_data_transform)
test_dataset = CustomImageDataset(test_df, transforms=test_data_transform)

# Load the train and test datasets

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

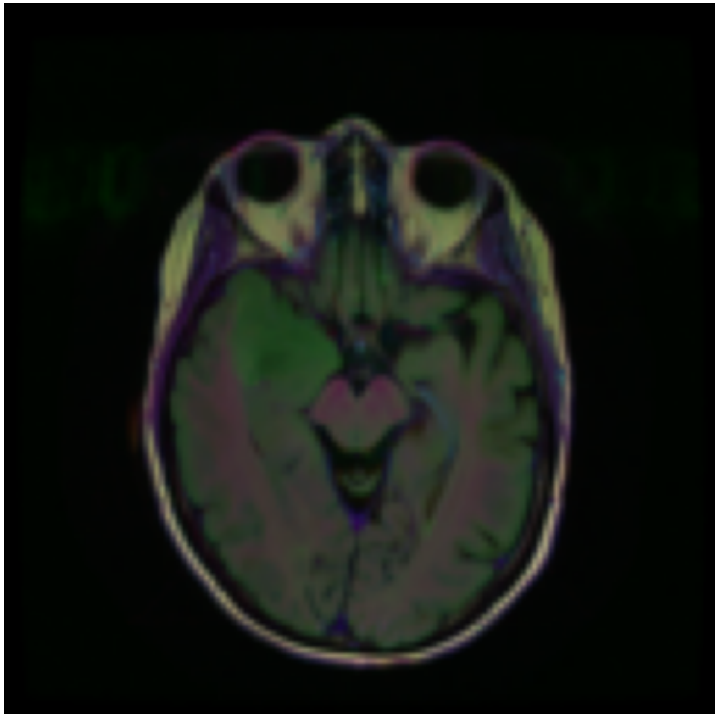
In [21]:
```
plt.figure(figsize=(6,4))
plt.bar(["Train", "Test"], [len(train_dataset), len(test_dataset)])
plt.ylabel('Size')
plt.title('Sizes of the Train and Test Datasets')
plt.show()
```

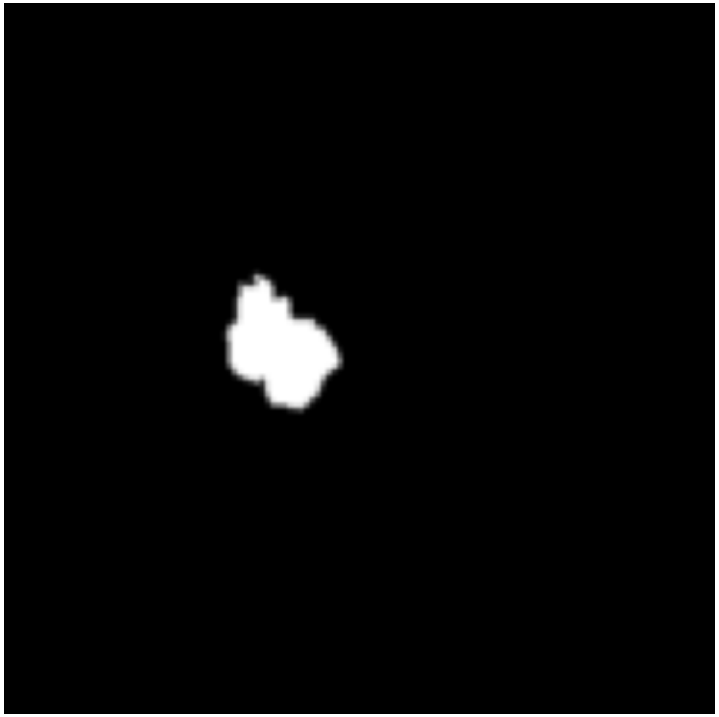


In [22]:
```
# Display sample images from the dataset
# np.random.seed(42)
incident = np.random.randint(0, len(train_dataset))

image, mask = train_dataset[incident]

plt.imshow(image.permute(1, 2, 0))
plt.axis("off")
plt.show()

plt.imshow(mask.permute(1, 2, 0), cmap="gray")
plt.axis("off")
plt.show()
```



In [23]:
```
# Display sample images from the dataset
# np.random.seed(42)
incident = np.random.randint(0, len(test_dataset))

image, mask = test_dataset[incident]

plt.imshow(image.permute(1, 2, 0))
plt.axis("off")
plt.show()

plt.imshow(mask.permute(1, 2, 0), cmap="gray")
plt.axis("off")
plt.show()
```





In [24]:
```
# Validate the train_loader

image, mask = next(iter(train_loader))
image.shape, mask.shape
```

Out[24]:
```
(torch.Size([16, 3, 128, 128]), torch.Size([16, 1, 128, 128]))
```

Deep Learning Model

Use U-Net architechture for the deep learning model

In [25]:
```
# Model building, inspired by the U-Net from Kaggle.

class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""
```

```
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )

    def forward(self, x):
        return self.double_conv(x)


class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2), DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)


class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()

        self.up = nn.ConvTranspose2d(
            in_channels // 2, in_channels // 2, kernel_size=2, stride=2
        )
        self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
```

```
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = nn.functional.pad(
            x1, [diffX // 2, diffX - diffX // 2, diffY // 2, diffY - diffY // 2]
        )

        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)


class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, 1)

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


class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        self.n_channels = 3
        self.n_classes = 1
        self.inc = DoubleConv(3, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        self.down4 = Down(512, 512)
        self.up1 = Up(1024, 256)
        self.up2 = Up(512, 128)
        self.up3 = Up(256, 64)
        self.up4 = Up(128, 64)
        self.outc = OutConv(64, 1)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
```

```
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits


# Create the UNet model
model = UNet().to(device)
```

Training Image Dataset with Tumor

In [26]:
```
# Define the loss function and optimizer

criterion = nn.BCEWithLogitsLoss()  # Suitable for binary segmentation
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [27]:
```
# Define the loss function

class DiceLoss(nn.Module):
    def __init__(self, smooth=1e-6):
        super(DiceLoss, self).__init__()
        self.smooth = smooth

    def forward(self, predictions, targets):
        predictions = predictions.sigmoid()  # Apply sigmoid to clamp between 0 and 1
        intersection = (
            predictions * targets
        ).sum()  # Element-wise multiplication and sum
        dice = (2.0 * intersection + self.smooth) / (
            predictions.sum() + targets.sum() + self.smooth
        )
        return 1 - dice

dice_loss = DiceLoss()
```

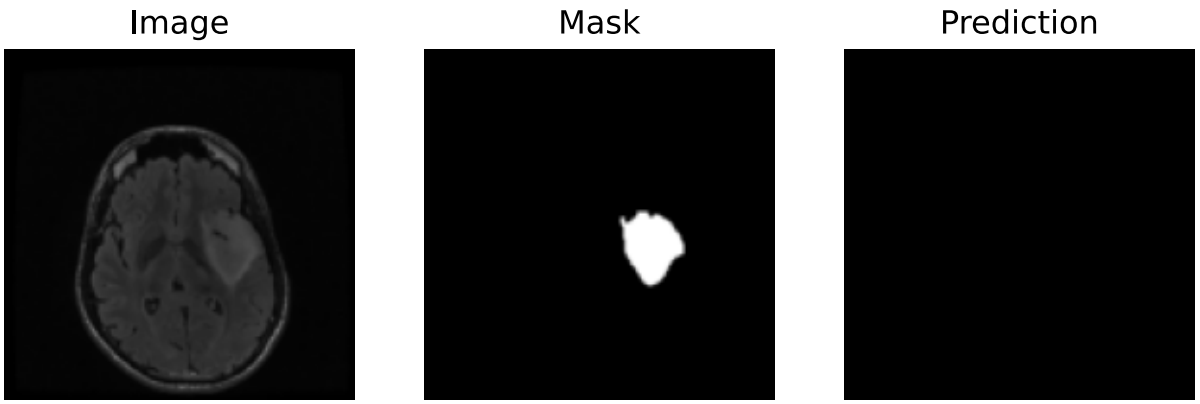
In [28]:
```
def training(train_loader, test_loader, model, num_epochs = 50):

    torch.cuda.empty_cache()

    # torch.manual_seed(42)
    train_loss_history = []
    val_loss_history = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for images, masks in train_loader:
            optimizer.zero_grad()

            images = images.to(device)
            masks = masks.to(device)

            outputs = model(images)
            # Resize masks to match the size of outputs
            # masks = torch.nn.functional.interpolate(masks, size=outputs.size()[2:], mode='bilinear', align_corners=False).to

            loss = dice_loss(outputs, masks)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)

        epoch_loss = running_loss / len(train_loader.dataset)
        train_loss_history.append(epoch_loss)
        print(f"Epoch {epoch+1}/{num_epochs}, Training Loss: {epoch_loss:.4f}")



        model.eval()
        with torch.no_grad():

            testing_loss = 0.0

            for images, masks in test_loader:
```

```
            images = images.to(device)
            masks = masks.to(device)
            outputs = model(images)
            preds = torch.sigmoid(outputs)
            preds = preds > 0.5  # Apply threshold

            loss = dice_loss(outputs, masks)
            testing_loss += loss.item() * images.size(0)

        val_loss = testing_loss / len(test_loader.dataset)
        val_loss_history.append(val_loss)

        print(f"Epoch {epoch+1}/{num_epochs}, Validation Loss: {val_loss:.4f}")

        if epoch % 5 == 0:

            # Visualization code (for example purposes, visualize the first batch)

            plot_side_by_side(images[0].cpu().permute(1,2,0), masks[0].cpu().permute(1,2,0), preds[0].cpu().numpy().squeez

            # plt.figure(figsize=(10, 5))
            # plt.subplot(1, 3, 1)
            # plt.imshow(transforms.ToPILImage()(images[0].cpu()))
            # plt.title("Input Image")
            # plt.axis("off")
#
            # plt.subplot(1, 3, 2)
            # plt.imshow(transforms.ToPILImage()(masks[0].cpu()), cmap="gray")
            # plt.title("Ground Truth Mask")
            # plt.axis("off")
#
            # plt.subplot(1, 3, 3)
            # plt.imshow(preds[0].cpu().numpy().squeeze(), cmap="gray")
            # plt.title("Predicted Mask")
            # plt.axis("off")
            #
            # plt.show()

            #plot_model_results(images[0], masks[0], preds[0].squeeze(0))
```

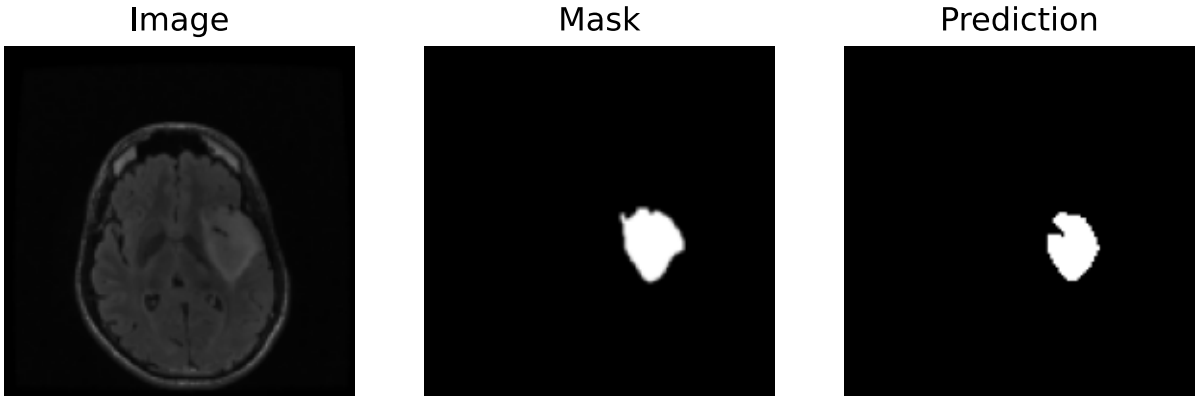
```
    return model, train_loss_history, val_loss_history
```

In [29]:
```
model, train_loss_history, val_loss_history = training(train_loader, test_loader, model, num_epochs=50)
```

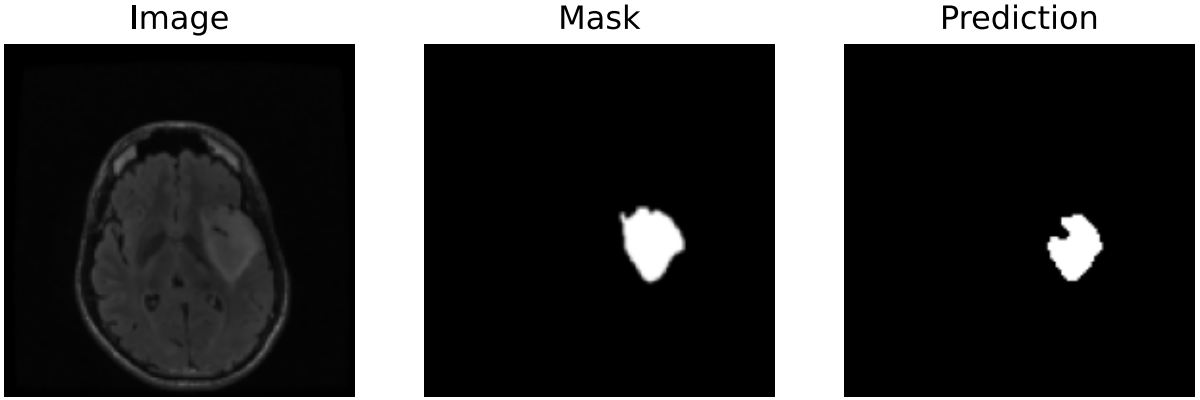
```
Epoch 1/50, Training Loss: 0.7180
Epoch 1/50, Validation Loss: 0.5379
```



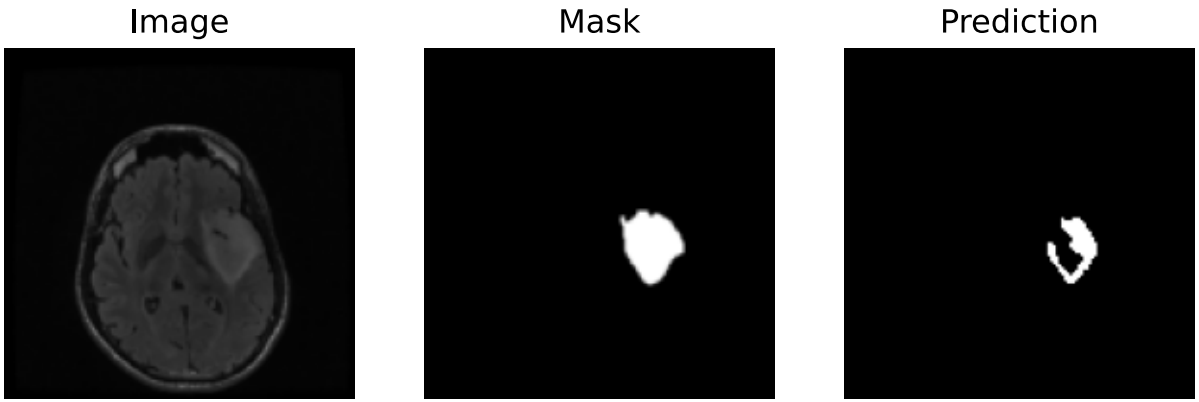

```
Epoch 2/50, Training Loss: 0.3962
Epoch 2/50, Validation Loss: 0.3344
Epoch 3/50, Training Loss: 0.3000
Epoch 3/50, Validation Loss: 0.3536
Epoch 4/50, Training Loss: 0.2795
Epoch 4/50, Validation Loss: 0.2867
Epoch 5/50, Training Loss: 0.2583
Epoch 5/50, Validation Loss: 0.2739
Epoch 6/50, Training Loss: 0.2453
Epoch 6/50, Validation Loss: 0.2364
```



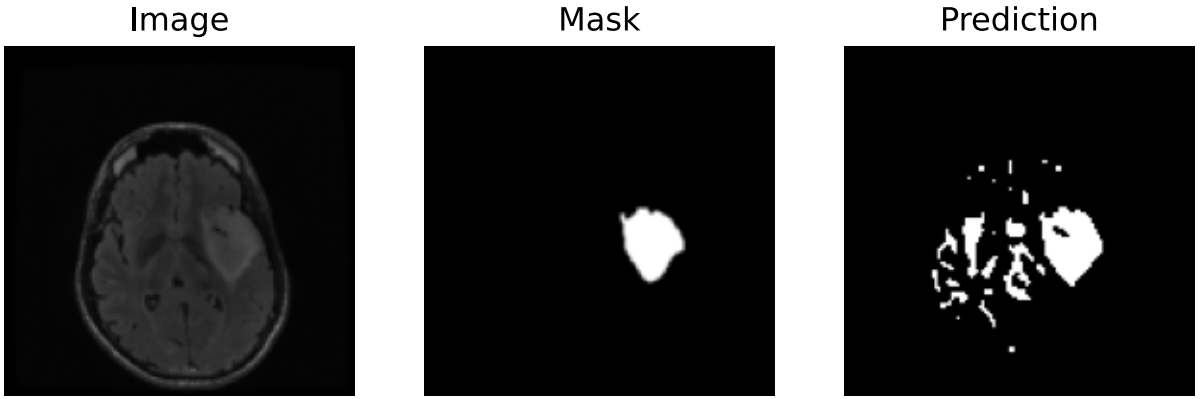

```
Epoch 7/50, Training Loss: 0.2429
Epoch 7/50, Validation Loss: 0.2628
Epoch 8/50, Training Loss: 0.2473
Epoch 8/50, Validation Loss: 0.3137
Epoch 9/50, Training Loss: 0.2468
Epoch 9/50, Validation Loss: 0.2579
Epoch 10/50, Training Loss: 0.2263
Epoch 10/50, Validation Loss: 0.2192
Epoch 11/50, Training Loss: 0.2199
Epoch 11/50, Validation Loss: 0.2482
```



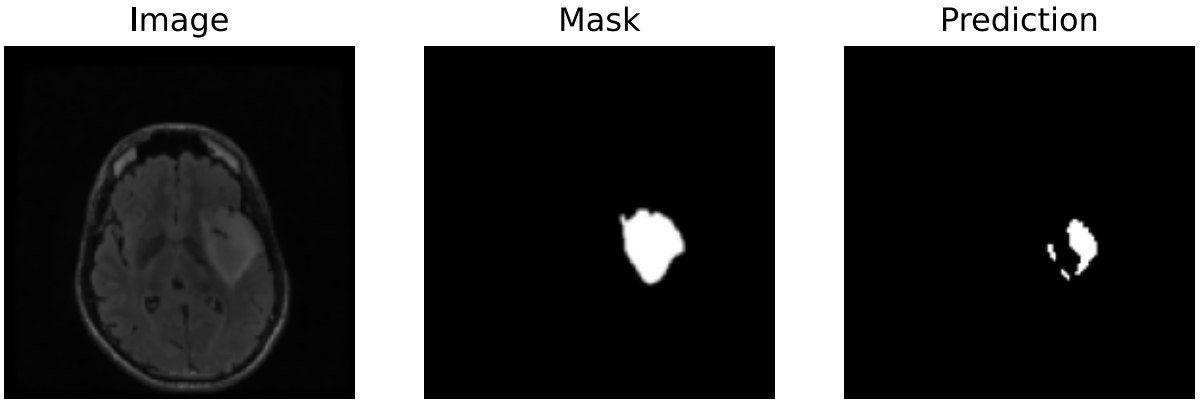
```
Epoch 12/50, Training Loss: 0.2175
Epoch 12/50, Validation Loss: 0.4148
Epoch 13/50, Training Loss: 0.2143
Epoch 13/50, Validation Loss: 0.2409
Epoch 14/50, Training Loss: 0.2156
Epoch 14/50, Validation Loss: 0.2626
Epoch 15/50, Training Loss: 0.2148
Epoch 15/50, Validation Loss: 0.2558
Epoch 16/50, Training Loss: 0.2228
Epoch 16/50, Validation Loss: 0.2250
```



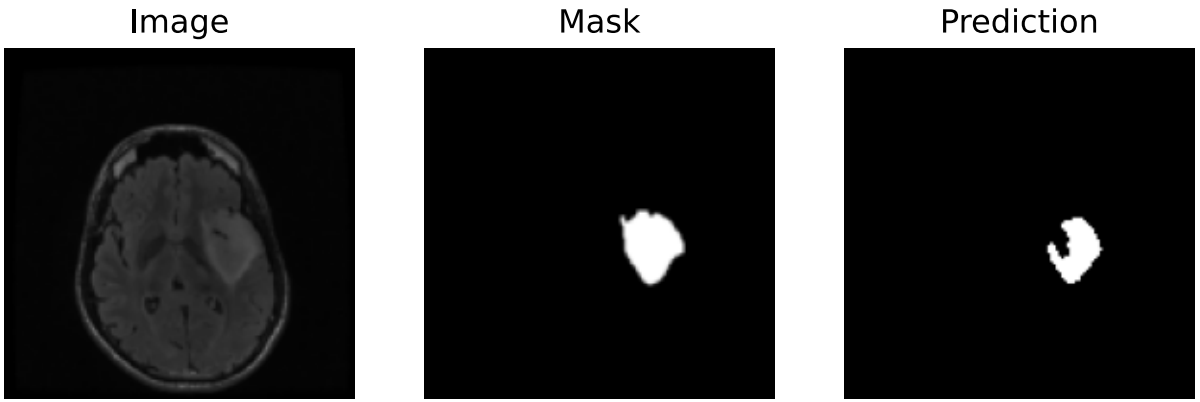

```
Epoch 17/50, Training Loss: 0.2152
Epoch 17/50, Validation Loss: 0.2371
Epoch 18/50, Training Loss: 0.2010
Epoch 18/50, Validation Loss: 0.2518
Epoch 19/50, Training Loss: 0.1980
Epoch 19/50, Validation Loss: 0.2049
Epoch 20/50, Training Loss: 0.1930
Epoch 20/50, Validation Loss: 0.2243
Epoch 21/50, Training Loss: 0.2029
Epoch 21/50, Validation Loss: 0.2541
```



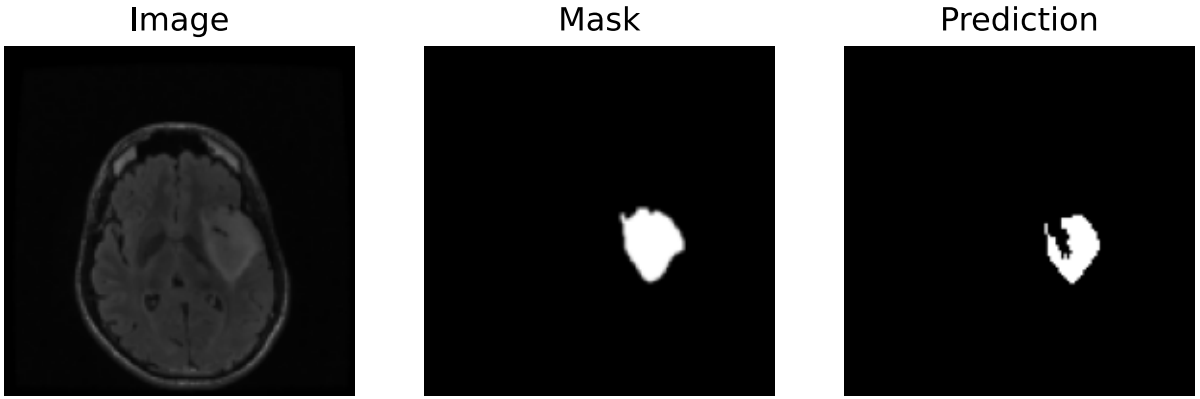

```
Epoch 22/50, Training Loss: 0.1953
Epoch 22/50, Validation Loss: 0.1979
Epoch 23/50, Training Loss: 0.1842
Epoch 23/50, Validation Loss: 0.1995
Epoch 24/50, Training Loss: 0.1768
Epoch 24/50, Validation Loss: 0.1915
Epoch 25/50, Training Loss: 0.1739
Epoch 25/50, Validation Loss: 0.1811
Epoch 26/50, Training Loss: 0.1751
Epoch 26/50, Validation Loss: 0.2046
```



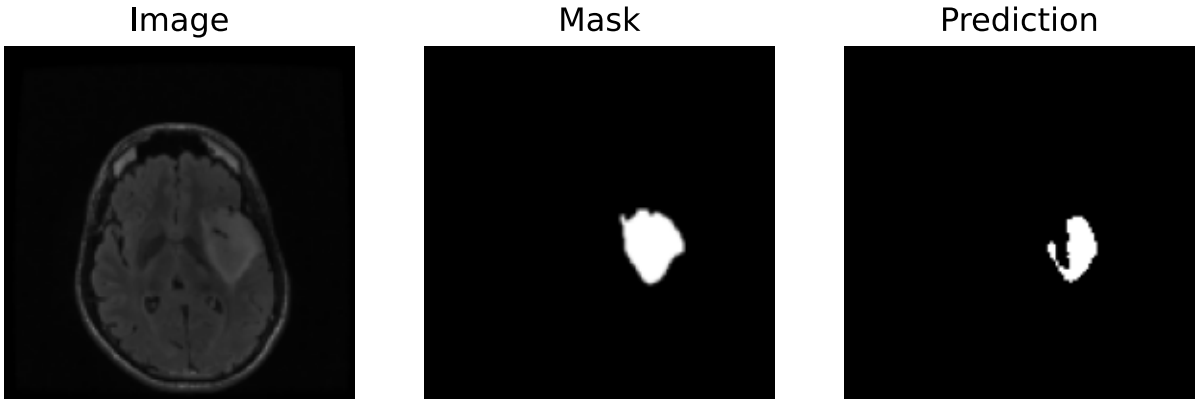
```
Epoch 27/50, Training Loss: 0.1881
Epoch 27/50, Validation Loss: 0.1907
Epoch 28/50, Training Loss: 0.1842
Epoch 28/50, Validation Loss: 0.2010
Epoch 29/50, Training Loss: 0.1772
Epoch 29/50, Validation Loss: 0.1961
Epoch 30/50, Training Loss: 0.1722
Epoch 30/50, Validation Loss: 0.2081
Epoch 31/50, Training Loss: 0.1741
Epoch 31/50, Validation Loss: 0.1891
```



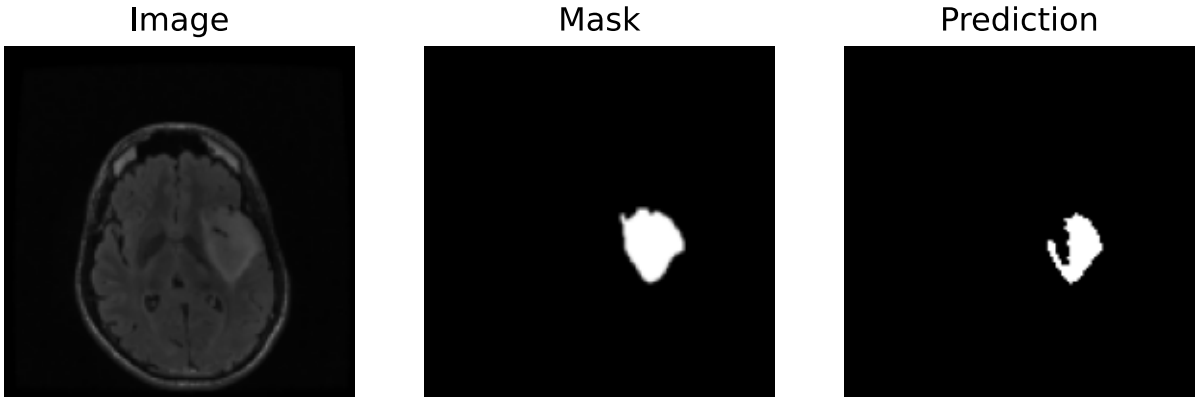

```
Epoch 32/50, Training Loss: 0.1702
Epoch 32/50, Validation Loss: 0.1812
Epoch 33/50, Training Loss: 0.1765
Epoch 33/50, Validation Loss: 0.1907
Epoch 34/50, Training Loss: 0.1699
Epoch 34/50, Validation Loss: 0.1901
Epoch 35/50, Training Loss: 0.1688
Epoch 35/50, Validation Loss: 0.1809
Epoch 36/50, Training Loss: 0.1634
Epoch 36/50, Validation Loss: 0.1803
```




```
Epoch 37/50, Training Loss: 0.1593
Epoch 37/50, Validation Loss: 0.1693
Epoch 38/50, Training Loss: 0.1636
Epoch 38/50, Validation Loss: 0.1733
Epoch 39/50, Training Loss: 0.1654
Epoch 39/50, Validation Loss: 0.1677
Epoch 40/50, Training Loss: 0.1588
Epoch 40/50, Validation Loss: 0.2002
Epoch 41/50, Training Loss: 0.1711
Epoch 41/50, Validation Loss: 0.2128
```



```
Epoch 42/50, Training Loss: 0.1706
Epoch 42/50, Validation Loss: 0.2007
Epoch 43/50, Training Loss: 0.1580
Epoch 43/50, Validation Loss: 0.1719
Epoch 44/50, Training Loss: 0.1616
Epoch 44/50, Validation Loss: 0.1935
Epoch 45/50, Training Loss: 0.1658
Epoch 45/50, Validation Loss: 0.1693
Epoch 46/50, Training Loss: 0.1535
Epoch 46/50, Validation Loss: 0.1439
```




```
Epoch 47/50, Training Loss: 0.1422
Epoch 47/50, Validation Loss: 0.1511
Epoch 48/50, Training Loss: 0.1430
Epoch 48/50, Validation Loss: 0.1587
Epoch 49/50, Training Loss: 0.1518
Epoch 49/50, Validation Loss: 0.1712
Epoch 50/50, Training Loss: 0.1628
Epoch 50/50, Validation Loss: 0.1807
```

In [30]:
```
torch.save(model.state_dict(), "unet.pth")
```

In [31]:
```
model = UNet().to(device)  # Replace with your actual model architecture
model.load_state_dict(torch.load("unet.pth"))
```

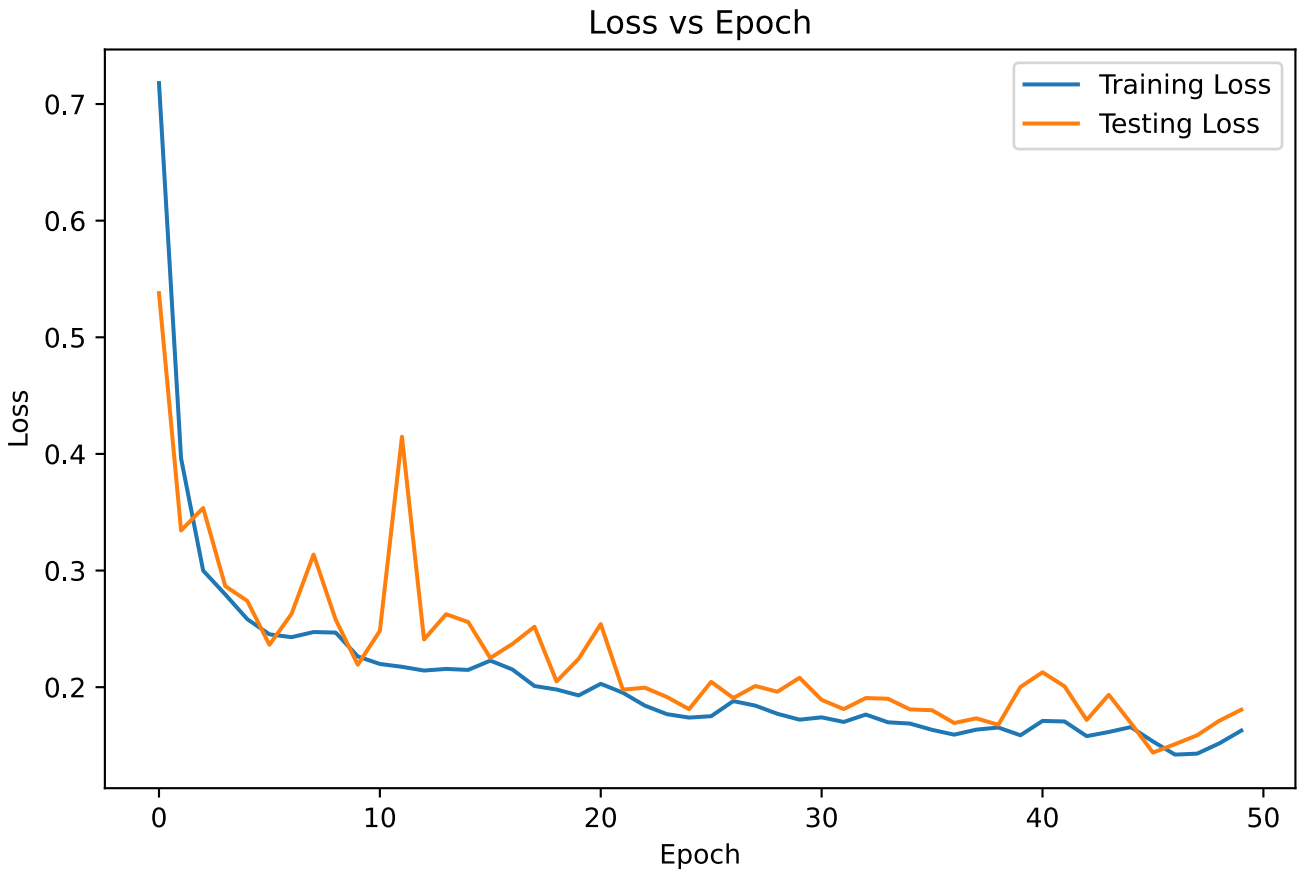
Out[31]:
```
<All keys matched successfully>
```

Evaluation

The ground truth mask (depicted by the green contour) and the predicted mask (represented by the red contour) are overlaid on the original image to visualize the evaluation.

In [32]:
```
def pred_plots(df, indices):

    fig, axs = plt.subplots(4, 4, figsize=(10, 10))

    for i, index in enumerate(indices):
        image = Image.open(df.iloc[index]["image_path"])
        mask = Image.open(df.iloc[index]["mask_path"])

        ax = axs[i // 4, i % 4]
        overlay = overlay_mask(image, mask)

        output = model(test_data_transform(image).unsqueeze(0).to(device))
        output = torch.sigmoid(output)
        output = output.squeeze().cpu().detach().numpy()

        overlay = overlay_output(overlay, output)

        ax.imshow(overlay)

        # ax.imshow(image)
        # ax.imshow(mask, cmap="jet", alpha=0.5)
        ax.axis("off")

        ax.set_title(f"Index: {index}")

    plt.suptitle("Green Contour: Ground Truth Mask\nRed Contour: Predicted Mask")

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

In [33]:
```
# Plot the loss history with epoch

plot_losses(train_loss_history, val_loss_history)
```

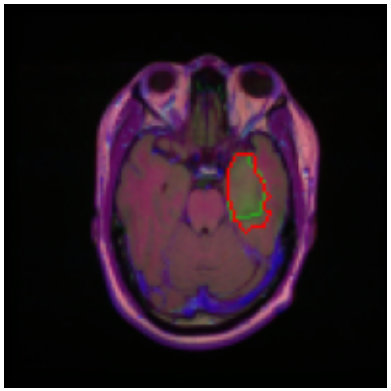


In [34]:
```
# Random pick 16 images from the `df_mask` and make image plots with the predicted mask

# Green contour presents the ground truth mask
# Red contour presents the predicted mask

# np.random.seed(42)
indices = np.random.randint(0, len(df_mask), 16)
pred_plots(df_mask, indices)
```

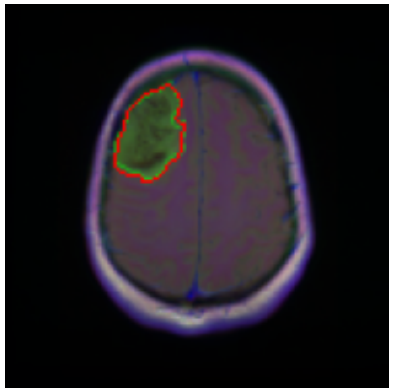
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Green Contour: Ground Truth Mask
Red Contour: Predicted Mask

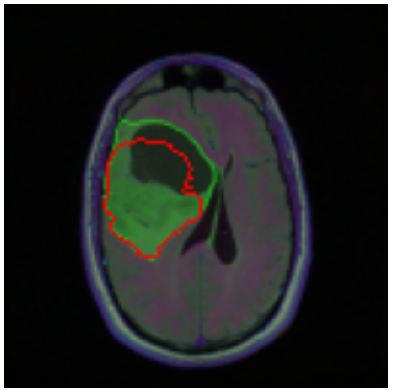
Index: 1171



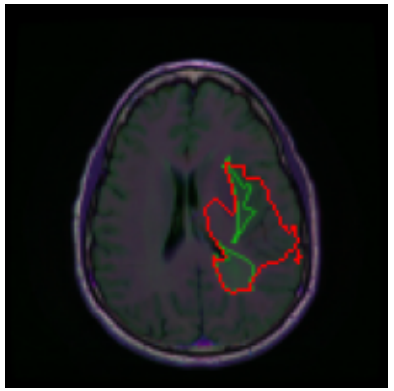
Index: 1073



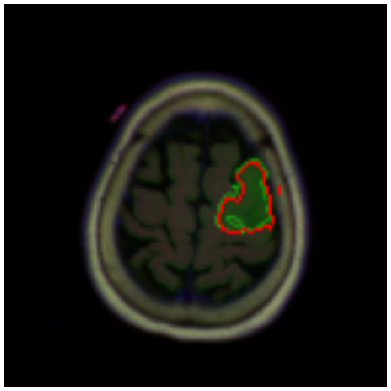
Index: 808



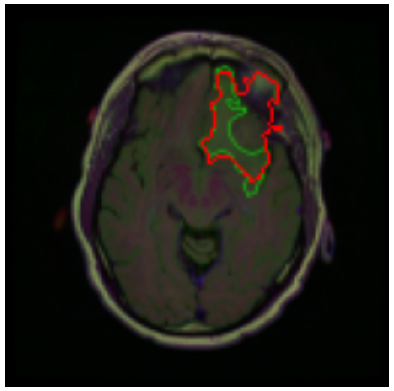
Index: 683



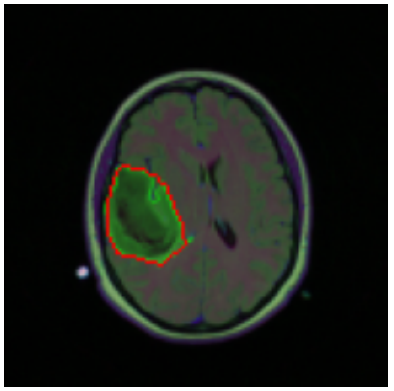
Index: 711



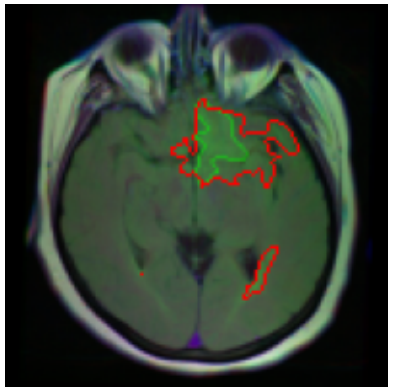
Index: 547



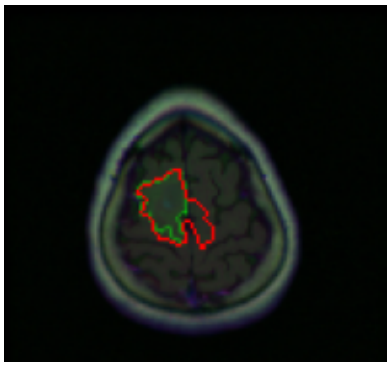
Index: 839



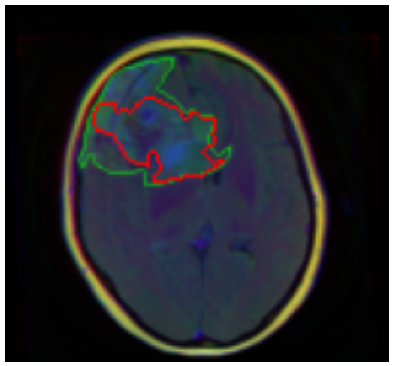
Index: 716



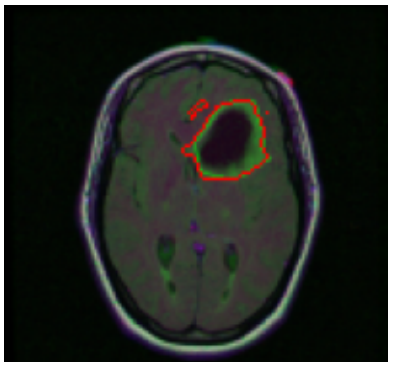
Index: 310



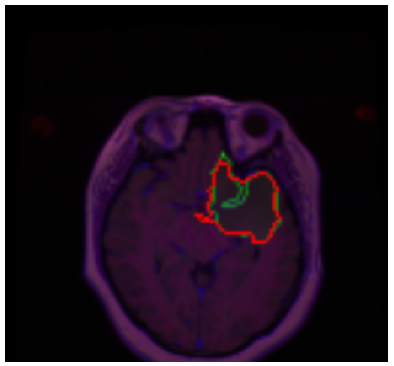
Index: 44

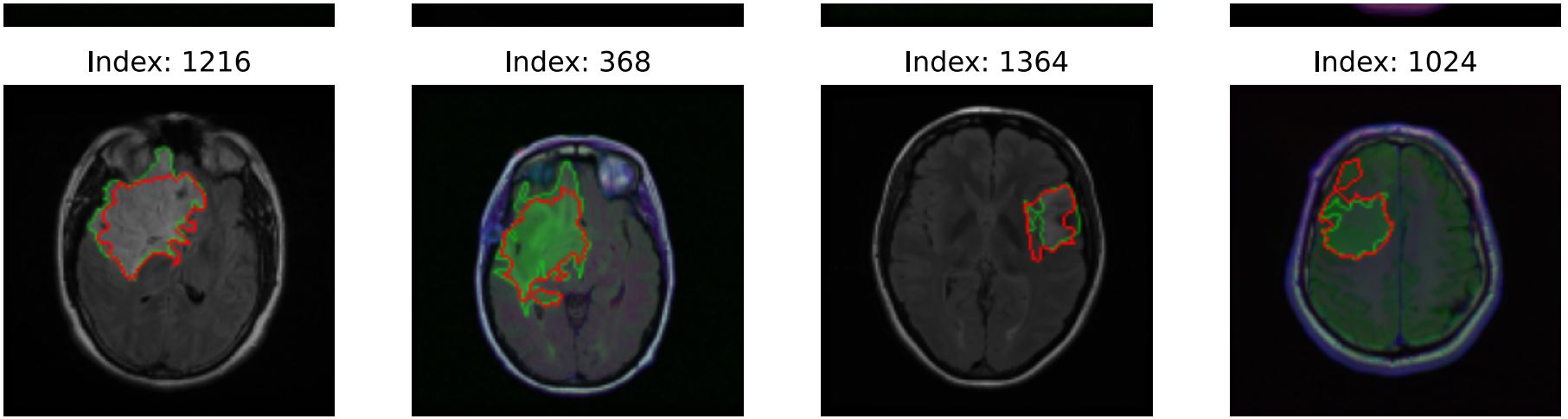


Index: 201



Index: 1310





Discussion

In this project, a deep learning-based method was developed to segment the cancerous regions of brain tumors. The U-Net deep learning architecture, widely recognized for its effectiveness in bioimage analysis, was implemented for the segmentation task. The model demonstrated promising results, with a training loss rate of 0.1628 and a validation loss rate of 0.1807 after 50 epochs. These results suggest that the algorithm is capable of generating accurate segmentations. The model was trained in approximately 60 minutes on a GPU.

However, there is potential for further optimization of the segmentation accuracy through improved image preprocessing steps. One approach could be to fine-tune the normalization parameters for the images to minimize variability. Implementing noise reduction and contrast enhancement could also enhance the quality of the MRI images, making tumor regions more distinguishable from normal tissue. Additionally, the use of a pretrained model for skull stripping during image preprocessing could be explored. Due to the project's time constraints, a universal image processing method that could unbiasedly preprocess all 1373 images in the dataset was not identified.

The choice of loss function is another crucial aspect of deep learning programming. Exploring different loss functions could potentially improve the model's performance.

The original dataset also includes images with blank masks (indicating brains without tumors). To simplify this course project, I only used images known to have masks. It could be intriguing to incorporate images without tumors into the image preprocessing to test if the algorithm can accurately handle images without tumors.

References

Mateusz Buda, AshirbaniSaha, Maciej A. Mazurowski "Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm." Computers in Biology and Medicine, 2019.

https://www.kaggle.com/code/truthisneverlinear/attention-u-net-pytorch

The code in this Jupyter Notebook was developed with assistance from GitHub Copilot. Initially, I wrote the code, and then GitHub Copilot provided revisions and improvements.

For more information about the dataset: https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation