Coursework 2: Image segmentation

In this coursework you will develop and train a convolutional neural network for brain tumour image segmentation. Please read both the text and the code in this notebook to get an idea what you are expected to implement. Pay attention to the missing code blocks that look like this:

```
### Insert your code ###
...
### End of your code ###
```

What to do?

- Complete and run the code using `jupyter-lab` or `jupyter-notebook` to get the results.
- Export (File | Save and Export Notebook As...) the notebook as a PDF file, which contains your code, results and answers, and upload the PDF file onto Scientia.
- Instead of clicking the Export button, you can also run the following command instead: `jupyter nbconvert coursework.ipynb --to pdf`
- If Jupyter complains about some problems in exporting, it is likely that pandoc (https://pandoc.org/installing.html) or latex is not installed, or their paths have not been included. You can install the relevant libraries and retry.
- If Jupyter-lab does not work for you at the end, you can use Google Colab to write the code and export the PDF file.

Dependencies

You need to install Jupyter-Lab (https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html) and other libraries used in this coursework, such as by running the command: `pip3 install [package_name]`

GPU resource

The coursework is developed to be able to run on CPU, as all images have been pre-processed to be 2D and of a smaller size, compared to original 3D volumes.

However, to save training time, you may want to use GPU. In that case, you can run this

notebook on Google Colab. On Google Colab, go to the menu, Runtime - Change runtime type, and select **GPU** as the hardware acceleartor. At the end, please still export everything and submit as a PDF file on Scientia.

```
# Import libraries
# These libraries should be sufficient for this tutorial.
# However, if any other library is needed, please install by yourself.
import tarfile
import imageio
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
import numpy as np
import time
import os
import random
import matplotlib.pyplot as plt
from matplotlib import colors
```

1. Download and visualise the imaging dataset.

The dataset is curated from the brain imaging dataset in [Medical Decathlon Challenge](). To save the storage and reduce the computational cost for this tutorial, we extract 2D image slices from T1-Gd contrast enhanced 3D brain volumes and downsample the images.

The dataset consists of a training set and a test set. Each image is of dimension 120 x 120, with a corresponding label map of the same dimension. There are four number of classes in the label map:

- 0: background
- 1: edema
- 2: non-enhancing tumour
- 3: enhancing tumour

```
# Download the dataset
!wget https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz

# Unzip the '.tar.gz' file to the current directory
datafile = tarfile.open('Task01_BrainTumour_2D.tar.gz')
```

```
datafile.extractall()
datafile.close()
```

```
--2024-02-26 14:22:35--  https://www.dropbox.com/s/zmytk2yu284af6t/Task01_B
Resolving www.dropbox.com (www.dropbox.com)... 162.125.3.18, 2620:100:6018:
Connecting to www.dropbox.com (www.dropbox.com)|162.125.3.18|:443... connec
HTTP request sent, awaiting response... 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz [following]
--2024-02-26 14:22:35--  https://www.dropbox.com/s/raw/zmytk2yu284af6t/Task
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucd237398850fbe46a277323d2b4.dl.dropboxusercontent.com/cd
--2024-02-26 14:22:35--  https://ucd237398850fbe46a277323d2b4.dl.dropboxuse
Resolving ucd237398850fbe46a277323d2b4.dl.dropboxusercontent.com (ucd237398
Connecting to ucd237398850fbe46a277323d2b4.dl.dropboxusercontent.com (ucd23
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/CODJA_wywp2206cUNfJzCqVrTRIuympKcKdVa72jwIuKM_WIqdB
--2024-02-26 14:22:36--  https://ucd237398850fbe46a277323d2b4.dl.dropboxuse
Reusing existing connection to ucd237398850fbe46a277323d2b4.dl.dropboxuserc
HTTP request sent, awaiting response... 200 OK
Length: 9251149 (8.8M) [application/octet-stream]
Saving to: 'Task01_BrainTumour_2D.tar.gz.1'

Task01_BrainTumour_ 100%[===================>]   8.82M  --.-KB/s    in 0.1s

2024-02-26 14:22:37 (70.3 MB/s) - 'Task01_BrainTumour_2D.tar.gz.1' saved [9
```

Visualise a random set of 4 training images along with their label maps.

Suggested colour map for brain MR image:

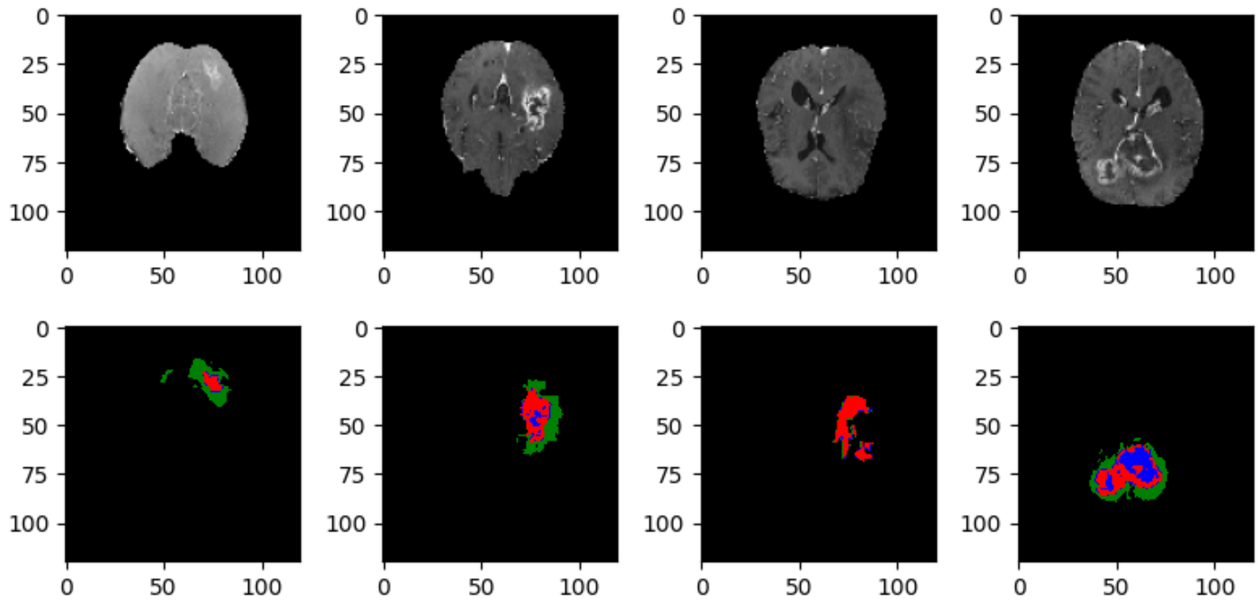
```
cmap = 'gray'
```

Suggested colour map for segmentation map:

```
cmap = colors.ListedColormap(['black', 'green', 'blue', 'red'])
```

```
### Insert your code ###

IMAGE_CMAP = "gray"
SEGMENTATION_CMAP = colors.ListedColormap(["black", "green", "blue", "red"])

path = "./Task01_BrainTumour_2D/training_images/"
training_imgs = os.listdir(path)
sample_imgs = random.sample(training_imgs, 4)
```

```
sample_imgs = random.sample(training_imgs, 4)

fig, axs = plt.subplots(2, 4, figsize=(8, 4))

for i in range(4):
  t_image = imageio.v2.imread(
      os.path.join("./Task01_BrainTumour_2D/training_images/", sample_imgs[i])
  )
  t_label = imageio.v2.imread(
      os.path.join("./Task01_BrainTumour_2D/training_labels/", sample_imgs[i])
  )

  axs[0,i].imshow(t_image, cmap=IMAGE_CMAP)
  axs[1,i].imshow(t_label, cmap=SEGMENTATION_CMAP)

plt.tight_layout()

### End of your code ###
```



2. Implement a dataset class.

It can read the imaging dataset and get items, pairs of images and label maps, as training batches.

```
def normalise_intensity(image, thres_roi=1.0):
    """ Normalise the image intensity by the mean and standard deviation """
    # ROI defines the image foreground
```

```
    val_l = np.percentile(image, thres_roi)
    roi = (image >= val_l)
    mu, sigma = np.mean(image[roi]), np.std(image[roi])
    eps = 1e-6
    image2 = (image - mu) / (sigma + eps)
    return image2


class BrainImageSet(Dataset):
    """ Brain image set """
    def __init__(self, image_path, label_path='', deploy=False):
        self.image_path = image_path
        self.deploy = deploy
        self.images = []
        self.labels = []

        image_names = sorted(os.listdir(image_path))
        for image_name in image_names:
            # Read the image
            image = imageio.v2.imread(os.path.join(image_path, image_name))
            self.images += [image]

            # Read the label map
            if not self.deploy:
                label_name = os.path.join(label_path, image_name)
                label = imageio.v2.imread(label_name)
                self.labels += [label]

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

    def __getitem__(self, idx):
        # Get an image and perform intensity normalisation
        # Dimension: XY
        image = normalise_intensity(self.images[idx])

        # Get its label map
        # Dimension: XY
        label = self.labels[idx]
        return image, label


    def get_random_batch(self, batch_size):
        # Get a batch of paired images and label maps
        # Dimension of images: NCXY
        # Dimension of labels: NXY
        images, labels = [], []

        ### Insert your code ###

        nums = range(len(self))
        batch_nums = random.sample(nums, batch_size)

        for n in batch_nums:
            image, label = self[n]
```

```
            image, label = self[n]
            image = np.expand_dims(image, axis=0)
            images.append(image)
            labels.append(label)

        images, labels = np.array(images), np.array(labels)

        ### End of your code ###
        return images, labels
```

3. Build a U-net architecture.

You will implement a U-net architecture. If you are not familiar with U-net, please read this paper:

[1] Olaf Ronneberger et al. [U-Net: Convolutional networks for biomedical image segmentation](). MICCAI, 2015.

For the first convolutional layer, you can start with 16 filters. We have implemented the encoder path. Please complete the decoder path.

```
""" U-net """
class UNet(nn.Module):
    def __init__(self, input_channel=1, output_channel=1, num_filter=16):
        super(UNet, self).__init__()

        # BatchNorm: by default during training this layer keeps running estima
        # of its computed mean and variance, which are then used for normalizat
        # during evaluation.

        # Encoder path
        n = num_filter  # 16
        self.conv1 = nn.Sequential(
            nn.Conv2d(input_channel, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n *= 2  # 32
        self.conv2 = nn.Sequential(
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )
```

```
        n *= 2  # 64
        self.conv3 = nn.Sequential(
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n *= 2  # 128
        self.conv4 = nn.Sequential(
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        # Decoder path
        ### Insert your code ###

        n //= 2  # 64
        self.trans1 = nn.ConvTranspose2d(n * 2, n, kernel_size=2, stride=2)
        self.decConv1 = nn.Sequential(
            nn.Conv2d(n * 2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n //= 2  # 32

        self.trans2 = nn.ConvTranspose2d(n * 2, n, kernel_size=2, stride=2)
        self.decConv2 = nn.Sequential(
            nn.Conv2d(n * 2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n //= 2  # 16
        self.trans3 = nn.ConvTranspose2d(n * 2, n, kernel_size=2, stride=2)
        self.decConv3 = nn.Sequential(
            nn.Conv2d(n * 2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
```

```
        )

        self.decConv4 = nn.Conv2d(n, output_channel, kernel_size=1)

        ### End of your code ###

    def forward(self, x):
        # Use the convolutional operators defined above to build the U-net
        # The encoder part is already done for you.
        # You need to complete the decoder part.
        # Encoder
        x = self.conv1(x)
        conv1_skip = x

        x = self.conv2(x)
        conv2_skip = x

        x = self.conv3(x)
        conv3_skip = x

        x = self.conv4(x)

        # Decoder
        ### Insert your code ###

        x = self.trans1(x)
        x = torch.cat([x, conv3_skip], dim=1)
        x = self.decConv1(x)

        x = self.trans2(x)
        x = torch.cat([x, conv2_skip], dim=1)
        x = self.decConv2(x)


        x = self.trans3(x)
        x = torch.cat([x, conv1_skip], dim=1)
        x = self.decConv3(x)

        x = self.decConv4(x)

        ### End of your code ###
        return x
```

4. Train the segmentation model.

```
# CUDA device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device: {0}'.format(device))

# Build the model
num_class = 4
model = UNet(input_channel=1, output_channel=num_class, num_filter=16)
```

```
model = model.to(device)
params = list(model.parameters())

model_dir = 'saved_models'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

# Optimizer
optimizer = optim.Adam(params, lr=1e-3)

# Segmentation loss
criterion = nn.CrossEntropyLoss()

# Datasets
train_set = BrainImageSet('Task01_BrainTumour_2D/training_images', 'Task01_Brain
test_set = BrainImageSet('Task01_BrainTumour_2D/test_images', 'Task01_BrainTumou

# Train the model
# Note: when you debug the model, you may reduce the number of iterations or bat
num_iter = 10000
train_batch_size = 16
eval_batch_size = 16
start = time.time()
for it in range(1, 1 + num_iter):
    # Set the modules in training mode, which will have effects on certain modul
    start_iter = time.time()
    model.train()

    # Get a batch of images and labels
    images, labels = train_set.get_random_batch(train_batch_size)
    images, labels = torch.from_numpy(images), torch.from_numpy(labels)
    images, labels = images.to(device, dtype=torch.float32), labels.to(device, d
    logits = model(images)

    # Perform optimisation and print out the training loss
    ### Insert your code ###

    # Perform backpropagation and optimisation
    loss_segmentation = criterion(logits, labels)
    optimizer.zero_grad()
    loss_segmentation.backward()
    optimizer.step()

    # Print training loss every 100 iterations / epochs
    if it % 100 == 0:
      print(f"Iteration [{it}], Training Loss: {loss_segmentation.item()}")

    ### End of your code ###

    # Evaluate
    if it % 100 == 0:
        model.eval()
        # Disabling gradient calculation during reference to reduce memory consu
        with torch.no_grad():
```

```
            # Evaluate on a batch of test images and print out the test loss
            ### Insert your code ###

            test_images, test_labels = test_set.get_random_batch(
                eval_batch_size
            )
            test_images = torch.from_numpy(test_images)
            test_labels = torch.from_numpy(test_labels)
            test_images = images.to(device, dtype=torch.float32)
            test_labels = labels.to(device, dtype=torch.long)
            test_logits = model(test_images)

            test_loss_segmentation = criterion(test_logits, test_labels)
            print(f"Iteration [{it}], Test Loss: {test_loss_segmentation.item()}

            ### End of your code ###

    # Save the model
    if it % 5000 == 0:
        torch.save(model.state_dict(), os.path.join(model_dir, 'model_{0}.pt'.fo
print('Training took {:.3f}s in total.'.format(time.time() - start))
```

```
Iteration [7100], Test Loss: 0.012177390977740288
Iteration [7200], Training Loss: 0.010905995033681393
Iteration [7200], Test Loss: 0.010273400694131851
Iteration [7300], Training Loss: 0.008096226491034031
Iteration [7300], Test Loss: 0.007243449334055185
Iteration [7400], Training Loss: 0.007855787873268127
Iteration [7400], Test Loss: 0.007591250352561474
Iteration [7500], Training Loss: 0.00887507013976574
Iteration [7500], Test Loss: 0.008260035887360573
Iteration [7600], Training Loss: 0.010916078463196754
Iteration [7600], Test Loss: 0.010431107133626938
Iteration [7700], Training Loss: 0.00791824609041214
Iteration [7700], Test Loss: 0.00718121649697423
Iteration [7800], Training Loss: 0.011470519006252289
Iteration [7800], Test Loss: 0.01064845360815525
Iteration [7900], Training Loss: 0.006690581329166889
Iteration [7900], Test Loss: 0.0062018828466534615
Iteration [8000], Training Loss: 0.013255341909825802
Iteration [8000], Test Loss: 0.012530828826129436
Iteration [8100], Training Loss: 0.008013137616217136
Iteration [8100], Test Loss: 0.007665497250854969
Iteration [8200], Training Loss: 0.007537941914051771
Iteration [8200], Test Loss: 0.006351415999233723
Iteration [8300], Training Loss: 0.007458283565938473
Iteration [8300], Test Loss: 0.00689641572535038
Iteration [8400], Training Loss: 0.01091840025037527
Iteration [8400], Test Loss: 0.010115638375282288
Iteration [8500], Training Loss: 0.0072942087426781654
Iteration [8500], Test Loss: 0.006879638414829969
Iteration [8600], Training Loss: 0.011648663319647312
Iteration [8600], Test Loss: 0.010914870537817478
Iteration [8700], Training Loss: 0.009938712231814861
Iteration [8700], Test Loss: 0.009731574915349483
Iteration [8800], Training Loss: 0.0059462860226631165
Iteration [8800], Test Loss: 0.00491725979372859
Iteration [8900], Training Loss: 0.009042593650519848
```

```
Iteration [8900], Test Loss: 0.00841287150979042
Iteration [9000], Training Loss: 0.00895696971565485
Iteration [9000], Test Loss: 0.008536763489246368
Iteration [9100], Training Loss: 0.007622076664119959
Iteration [9100], Test Loss: 0.007021469064056873
Iteration [9200], Training Loss: 0.013709107413887978
Iteration [9200], Test Loss: 0.011984104290604591
Iteration [9300], Training Loss: 0.005870596040040255
Iteration [9300], Test Loss: 0.005553945899009705
Iteration [9400], Training Loss: 0.006853488739579916
Iteration [9400], Test Loss: 0.006246726028621197
Iteration [9500], Training Loss: 0.007025748956948519
Iteration [9500], Test Loss: 0.0069920071400702
Iteration [9600], Training Loss: 0.010276216082274914
Iteration [9600], Test Loss: 0.009721535257995129
Iteration [9700], Training Loss: 0.009051240980625153
Iteration [9700], Test Loss: 0.008398638106882572
Iteration [9800], Training Loss: 0.009543166495859623
Iteration [9800], Test Loss: 0.009091171436011791
Iteration [9900], Training Loss: 0.00684011448174715
Iteration [9900], Test Loss: 0.006181488744914532
Iteration [10000], Training Loss: 0.0063770245760679245
Iteration [10000], Test Loss: 0.005908176768571138
Training took 331.399s in total
```

Full Output:

Device: cuda

Iteration [100], Training Loss: 0.45683836936950684

Iteration [100], Test Loss: 0.4459705054759979

Iteration [200], Training Loss: 0.21147119998931885

Iteration [200], Test Loss: 0.21752020716667175

Iteration [300], Training Loss: 0.12818099558353424

Iteration [300], Test Loss: 0.13360750675201416

Iteration [400], Training Loss: 0.09219790250062943

Iteration [400], Test Loss: 0.08921840786933899

Iteration [500], Training Loss: 0.058779530227184296

Iteration [500], Test Loss: 0.06271056085824966

Iteration [600], Training Loss: 0.07287901639938354

Iteration [600], Test Loss: 0.10345973819494247

Iteration [700], Training Loss: 0.07269126176834106

Iteration [700], Test Loss: 0.07619970291852951

Iteration [800], Training Loss: 0.057208627462387085

Iteration [800], Test Loss: 0.059080082923173904

Iteration [900], Training Loss: 0.061121612787246704

Iteration [900], Test Loss: 0.07950388640165329

Iteration [1000], Training Loss: 0.049168117344379425

Iteration [1000], Test Loss: 0.049158740788698196

Iteration [1100], Training Loss: 0.0391080267727375

Iteration [1100], Test Loss: 0.04021170735359192

Iteration [1200], Training Loss: 0.05022561922669411

Iteration [1200], Test Loss: 0.04611064866185188

Iteration [1300], Training Loss: 0.0460481122136116

Iteration [1300], Test Loss: 0.05266570299863815

Iteration [1400], Training Loss: 0.026371510699391365

Iteration [1400], Test Loss: 0.022829944267868996

Iteration [1500], Training Loss: 0.045412372797727585

Iteration [1500], Test Loss: 0.04366118088364601

Iteration [1600], Training Loss: 0.041420962661504745

Iteration [1600], Test Loss: 0.04802503436803818

Iteration [1700], Training Loss: 0.044234149158000946

Iteration [1700], Test Loss: 0.041484903544187546

Iteration [1800], Training Loss: 0.03582900017499924

Iteration [1800], Test Loss: 0.03317388519644737

Iteration [1900], Training Loss: 0.021971849724650383

Iteration [1900], Test Loss: 0.021574802696704865

Iteration [2000], Training Loss: 0.028142716735601425

Iteration [2000], Test Loss: 0.031229844316840172

Iteration [2100], Training Loss: 0.029840996488928795

Iteration [2100], Test Loss: 0.028730234131217003

Iteration [2200], Training Loss: 0.025411872193217278

Iteration [2200], Test Loss: 0.0239359512925148

Iteration [2300], Training Loss: 0.019643783569335938

Iteration [2300], Test Loss: 0.01906718499958515

Iteration [2400], Training Loss: 0.028899963945150375

Iteration [2400], Test Loss: 0.026648491621017456

Iteration [2500], Training Loss: 0.02847537212073803

Iteration [2500], Test Loss: 0.025562137365341187

Iteration [2600], Training Loss: 0.013416269794106483

Iteration [2600], Test Loss: 0.013158712536096573

Iteration [2700], Training Loss: 0.015232156030833721

Iteration [2700], Test Loss: 0.014376288279891014

Iteration [2800], Training Loss: 0.019305946305394173

Iteration [2800], Test Loss: 0.017563488334417343

Iteration [2900], Training Loss: 0.031292594969272614

Iteration [2900], Test Loss: 0.031880687922239304

Iteration [3000], Training Loss: 0.022699672728776932

Iteration [3000], Test Loss: 0.02253621816635132

Iteration [3100], Training Loss: 0.03152637928724289

Iteration [3100], Test Loss: 0.03041081689298153

Iteration [3200], Training Loss: 0.022866712883114815

Iteration [3200], Test Loss: 0.021187899634242058

Iteration [3300], Training Loss: 0.019755104556679726

Iteration [3300], Test Loss: 0.019400015473365784

Iteration [3400], Training Loss: 0.024823807179927826

Iteration [3400], Test Loss: 0.024043988436460495

Iteration [3500], Training Loss: 0.019840769469738007

Iteration [3500], Test Loss: 0.016804350540041924

Iteration [3600], Training Loss: 0.020125018432736397

Iteration [3600], Test Loss: 0.01921933889389038

Iteration [3700], Training Loss: 0.017322059720754623

Iteration [3700], Test Loss: 0.01703800819814205

Iteration [3800], Training Loss: 0.017584294080734253

Iteration [3800], Test Loss: 0.01711105927824974

Iteration [3900], Training Loss: 0.022121943533420563

Iteration [3900], Test Loss: 0.019108572974801064

Iteration [4000], Training Loss: 0.020981233566999435

Iteration [4000], Test Loss: 0.020125100389122963

Iteration [4100], Training Loss: 0.009977445006370544

Iteration [4100], Test Loss: 0.009295784868299961

Iteration [4200], Training Loss: 0.014105564914643764

Iteration [4200], Test Loss: 0.013643160462379456

Iteration [4300], Training Loss: 0.021360591053962708

Iteration [4300], Test Loss: 0.01917637698352337

Iteration [4400], Training Loss: 0.012657136656343937

Iteration [4400], Test Loss: 0.012178721837699413

Iteration [4500], Training Loss: 0.014616605825722218

Iteration [4500], Test Loss: 0.013960606418550014

Iteration [4600], Training Loss: 0.018278323113918304

Iteration [4600], Test Loss: 0.017474042251706123

Iteration [4700], Training Loss: 0.011666857637465

Iteration [4700], Test Loss: 0.011326338164508343

Iteration [4800], Training Loss: 0.013464334420859814

Iteration [4800], Test Loss: 0.013624496757984161

Iteration [4900], Training Loss: 0.016846144571900368

Iteration [4900], Test Loss: 0.016457775607705116

Iteration [5000], Training Loss: 0.019765542820096016

Iteration [5000], Test Loss: 0.017647847533226013

Iteration [5100], Training Loss: 0.01731441542506218

Iteration [5100], Test Loss: 0.01601111888885498

Iteration [5200], Training Loss: 0.015122732147574425

Iteration [5200], Test Loss: 0.014996283687651157

Iteration [5300], Training Loss: 0.01586892642080784

Iteration [5300], Test Loss: 0.014640050008893013

Iteration [5400], Training Loss: 0.008701161481440067

Iteration [5400], Test Loss: 0.008379785344004631

Iteration [5500], Training Loss: 0.009250923059880733

Iteration [5500], Test Loss: 0.009332973510026932

Iteration [5600], Training Loss: 0.003870565677061677

Iteration [5600], Test Loss: 0.0037502888590097427

Iteration [5700], Training Loss: 0.011731186881661415

Iteration [5700], Test Loss: 0.011316136457026005

Iteration [5800], Training Loss: 0.016105610877275467

Iteration [5800], Test Loss: 0.014852228574454784

Iteration [5900], Training Loss: 0.00760298129171133

Iteration [5900], Test Loss: 0.008067084476351738

Iteration [6000], Training Loss: 0.012823284603655338

Iteration [6000], Test Loss: 0.011432923376560211

Iteration [6100], Training Loss: 0.016060873866081238

Iteration [6100], Test Loss: 0.013875167816877365

Iteration [6200], Training Loss: 0.015723763033747673

Iteration [6200], Test Loss: 0.014197767712175846

Iteration [6300], Training Loss: 0.01503079105168581

Iteration [6300], Test Loss: 0.01468410063534975

Iteration [6400], Training Loss: 0.009217501617968082

Iteration [6400], Test Loss: 0.008811919018626213

Iteration [6500], Training Loss: 0.012432800605893135

Iteration [6500], Test Loss: 0.01177185121923685

Iteration [6600], Training Loss: 0.008689734153449535

Iteration [6600], Test Loss: 0.00821586325764656

Iteration [6700], Training Loss: 0.012376963160932064

Iteration [6700], Test Loss: 0.011505871079862118

Iteration [6800], Training Loss: 0.013555847108364105

Iteration [6800], Test Loss: 0.012041794136166573

Iteration [6900], Training Loss: 0.01013008039444685

Iteration [6900], Test Loss: 0.01005529798567295

- -

Iteration [7000], Training Loss: 0.010588366538286209

Iteration [7000], Test Loss: 0.009641710668802261

Iteration [7100], Training Loss: 0.012212621048092842

Iteration [7100], Test Loss: 0.012177390977740288

Iteration [7200], Training Loss: 0.010905995033681393

Iteration [7200], Test Loss: 0.010273400694131851

Iteration [7300], Training Loss: 0.008096226491034031

Iteration [7300], Test Loss: 0.007243449334055185

Iteration [7400], Training Loss: 0.007855787873268127

Iteration [7400], Test Loss: 0.007591250352561474

Iteration [7500], Training Loss: 0.00887507013976574

Iteration [7500], Test Loss: 0.008260035887360573

Iteration [7600], Training Loss: 0.010916078463196754

Iteration [7600], Test Loss: 0.010431107133626938

Iteration [7700], Training Loss: 0.00791824609041214

Iteration [7700], Test Loss: 0.00718121649697423

Iteration [7800], Training Loss: 0.011470519006252289

Iteration [7800], Test Loss: 0.01064845360815525

Iteration [7900], Training Loss: 0.006690581329166889

Iteration [7900], Test Loss: 0.0062018828466534615

Iteration [8000], Training Loss: 0.013255341909825802

Iteration [8000], Test Loss: 0.012530828826129436

Iteration [8100], Training Loss: 0.008013137616217136

Iteration [8100], Test Loss: 0.007665497250854969

Iteration [8200], Training Loss: 0.007537941914051771

Iteration [8200], Test Loss: 0.006351415999233723

Iteration [8300], Training Loss: 0.007458283565938473

Iteration [8300], Test Loss: 0.00689641572535038

Iteration [8400], Training Loss: 0.01091840025037527

Iteration [8400], Test Loss: 0.010115638375282288

Iteration [8500], Training Loss: 0.0072942087426781654

Iteration [8500], Test Loss: 0.006879638414829969

Iteration [8600], Training Loss: 0.011648663319647312

Iteration [8600], Test Loss: 0.010914870537817478

Iteration [8700], Training Loss: 0.009938712231814861

Iteration [8700], Test Loss: 0.009731574915349483

Iteration [8800], Training Loss: 0.0059462860226631165

Iteration [8800], Test Loss: 0.00491725979372859

Iteration [8900], Training Loss: 0.009042593650519848

Iteration [8900], Test Loss: 0.00841287150979042

Iteration [9000], Training Loss: 0.00895696971565485

Iteration [9000], Test Loss: 0.008536763489246368

Iteration [9100], Training Loss: 0.007622076664119959

Iteration [9100], Test Loss: 0.007021469064056873

Iteration [9200], Training Loss: 0.013709107413887978

Iteration [9200], Test Loss: 0.011984104290604591

Iteration [9300], Training Loss: 0.005870596040040255

Iteration [9300], Test Loss: 0.005553945899009705

Iteration [9400], Training Loss: 0.006853488739579916

Iteration [9400], Test Loss: 0.006246726028621197

Iteration [9500], Training Loss: 0.007025748956948519

Iteration [9500], Test Loss: 0.0069920071400702

Iteration [9600], Training Loss: 0.010276216082274914

Iteration [9600], Test Loss: 0.009721535257995129

Iteration [9700], Training Loss: 0.009051240980625153

Iteration [9700], Test Loss: 0.008398638106882572

Iteration [9800], Training Loss: 0.009543166495859623

Iteration [9800], Test Loss: 0.009091171436011791

Iteration [9900], Training Loss: 0.00684011448174715

Iteration [9900], Test Loss: 0.006181488744914532

Iteration [10000], Training Loss: 0.0063770245760679245

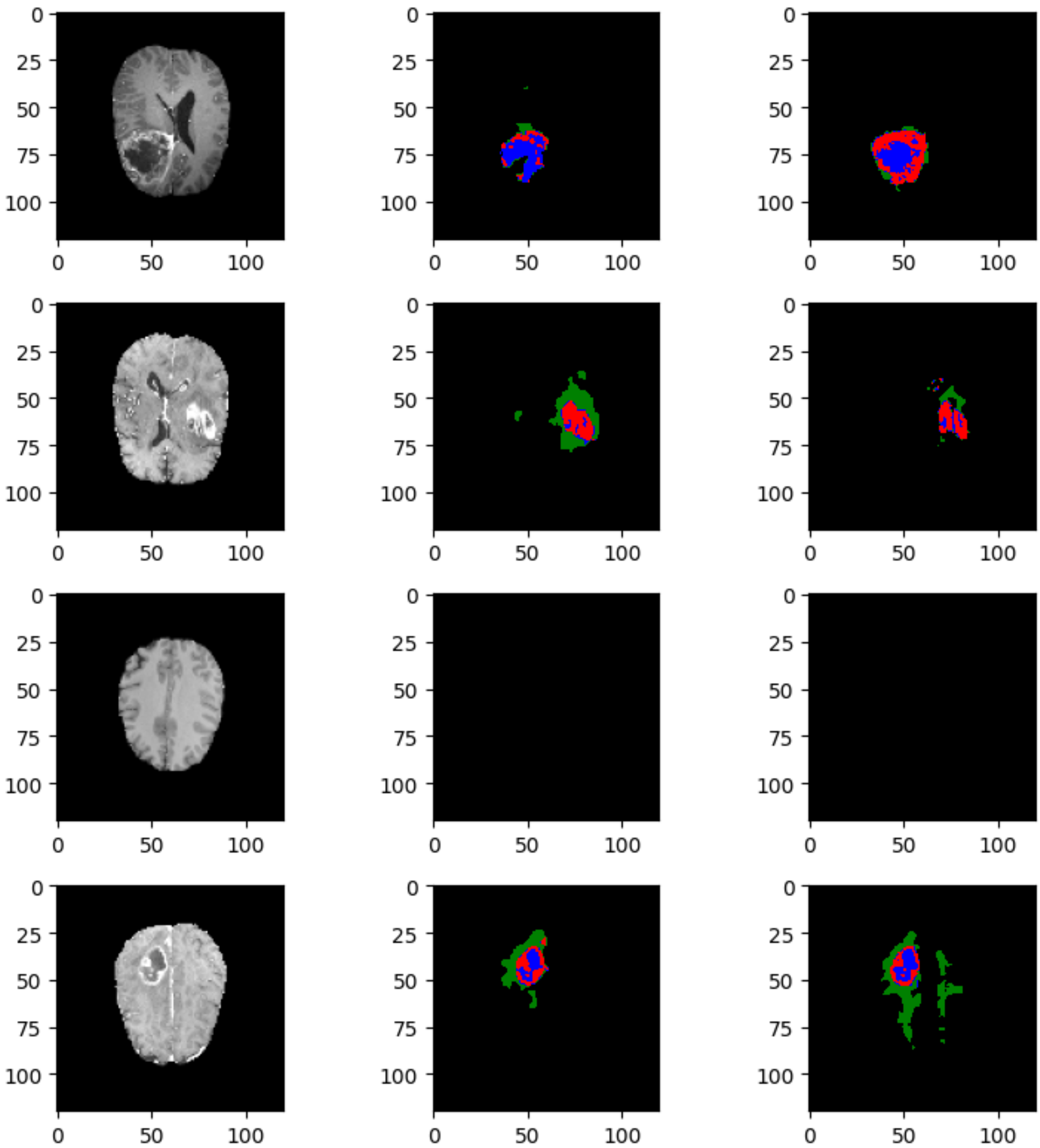
Iteration [10000], Test Loss: 0.0059081767685711138

Training took 331.399s in total.

5. Deploy the trained model to a random set of 4 test images and visualise the automated segmentation.

You can show the images as a 4 x 3 panel. Each row shows one example, with the 3 columns being the test image, automated segmentation and ground truth segmentation.

```
### Insert your code ###

# No need to load from saved model, since we can reuse variables from part 4
# Set model to evaluation mode
model.eval()

# Randomly sample 4 test images, from test_set for model training in part 4
test_sample, test_labels = test_set.get_random_batch(4)

# Deploy model on test sample
with torch.no_grad():
  test_sample = torch.from_numpy(test_sample)
  test_sample = test_sample.to(device, dtype=torch.float32)
  test_segmentation = model(test_sample)

fig, axs = plt.subplots(4, 3, figsize=(8, 8))

for i in range(4):
  t_image = test_sample[i].cpu()
  t_segmentation = test_segmentation[i].cpu()
  t_label = test_labels[i]

  # Apply softmax to each pixel
  probabilities = np.exp(t_segmentation) / np.exp(t_segmentation).sum(axis=0)
  # Determine maximum for each pixel, flattens: 4x120x120 => 120x120
  predicted_classes = np.argmax(probabilities, axis=0)

  axs[i,0].imshow(t_image[0], cmap=IMAGE_CMAP)
  axs[i,1].imshow(predicted_classes, cmap=SEGMENTATION_CMAP)
  axs[i,2].imshow(t_label, cmap=SEGMENTATION_CMAP)

plt.tight_layout()

### End of your code ###
```



6. Discussion. Does your trained model work well? How would you improve this model so it can be deployed to the real clinic?

The trained model is able to identify the instances of tumours fairly accurately, however, as shown in part 5, the model sometimes overlooks some regions of the tumour, or incorrectly identifies non-tumour regions as part of the tumour.

Some of this inaccuracy could be due to the simplified implementation of the U-Net architecture in the model. While it has allowed for quicker training, this simplified version might not be able to accurately learn features in the images. Therefore, one potential improvement would be to implement a more complex architecture, with more convolutional layers.

Another improvement involves enriching the training dataset. The current set only comprises a couple thousand images, which may be potentially limiting the accuracy of the trained model. Although using a more extensive dataset will prolong training time, the model's improved accuracy may be a worthwhile tradeoff for deployment in a real clinic.