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.

In [ ]:
```
# 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

In [ ]:
```
# 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()
```

```
--2023-03-03 01:22:24--  https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
Resolving www.dropbox.com (www.dropbox.com)... 162.125.2.18, 2620:100:6031:18::a27d:5112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.2.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz [following]
--2023-03-03 01:22:25--  https://www.dropbox.com/s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com/cd/0/inline/B3i2baiL24SngV6S-MHHzpr86oZ-DTIz3ZwkiON4Ggr0mRl3yEB60onCUk-2IVrnLKLIrYEQ2TQ9nBgIkfj7SNHU3xfN0iZhTEfcFjn2wOVFVGMlPasELqEA_AoCByd5nuWhzb46AEjmZkBJUxD2x7geQFiRfTu9_Fb44KH9fWC-3g/file# [following]
--2023-03-03 01:22:25--  https://uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com/cd/0/inline/B3i2baiL24SngV6S-MHHzpr86oZ-DTIz3ZwkiON4Ggr0mRl3yEB60onCUk-2IVrnLKLIrYEQ2TQ9nBgIkfj7SNHU3xfN0iZhTEfcFjn2wOVFVGMlPasELqEA_AoCByd5nuWhzb46AEjmZkBJUxD2x7geQFiRfTu9_Fb44KH9fWC-3g/file
Resolving uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com (uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com)... 162.125.81.15, 2620:100:6031:15::a27d:510f
Connecting to uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com (uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com)|162.125.81.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/B3hdlL8HuzOJrCeChKSt0gB7-VmpB0Bqc4jSMPcKgi8z4enlqCuNxkry4I3gY_61QE9abqNTHtXpF2TbsORhYzpTdeqGtT977_otNuOQCJneC-whlMjxo8OVwMhFN_RTYQcGCEPYMRTyiNf_JmScLAqLFpaKOawPLWDnmy-7sUNgMfx56i5Z6k4fBm0SoQ2OpMXANedXKpgKlHdSoogkfqzZ1k2I6H5gB8Ki38vKhE24wNZPV8Ds120NedyWSlmcZOHhMFt2VrPQa_HMiDM6_lXPQ1xoE-W0Qxbqlw1ijTikcnO9FOh-zq8DVk_Jv5B02xuHlo0nO1G3JwNzBFPCdanOiZP-D_wwONiSDjYvADxvAksSlWa6lyT66hmCpKz_vf2LKrpgBHIz_bhXyG2LR6VgvFxWkfpHUa34fVLXDWxtSQ/file [following]
--2023-03-03 01:22:26--  https://uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com/cd/0/inline2/B3hdlL8HuzOJrCeChKSt0gB7-VmpB0Bqc4jSMPcKgi8z4enlqCuNxkry4I3gY_61QE9abqNTHtXpF2TbsORhYzpTdeqGtT977_otNuOQCJneC-whlMjxo8OVwMhFN_RTYQcGCEPYMRTyiNf_JmScLAqLFpaKOawPLWDnmy-7sUNgMfx56i5Z6k4fBm0SoQ2OpMXANedXKpgKlHdSoogkfqzZ1k2I6H5gB8Ki38vKhE24wNZPV8Ds120NedyWSlmcZOHhMFt2VrPQa_HMiDM6_lXPQ1xoE-W0Qxbqlw1ijTikcnO9FOh-zq8DVk_Jv5B02xuHlo0nO1G3JwNzBFPCdanOiZP-D_wwONiSDjYvADxvAksSlWa6lyT66hmCpKz_vf2LKrpgBHIz_bhXyG2LR6VgvFxWkfpHUa34fVLXDWxtSQ/file
Reusing existing connection to uc07d24e75008b170a599146cc24.dl.dropboxusercontent.com:443.
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  11.6MB/s    in 0.8s

2023-03-03 01:22:27 (11.6 MB/s) - ‘Task01_BrainTumour_2D.tar.gz.1’ saved [9251149/9251149]
```

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

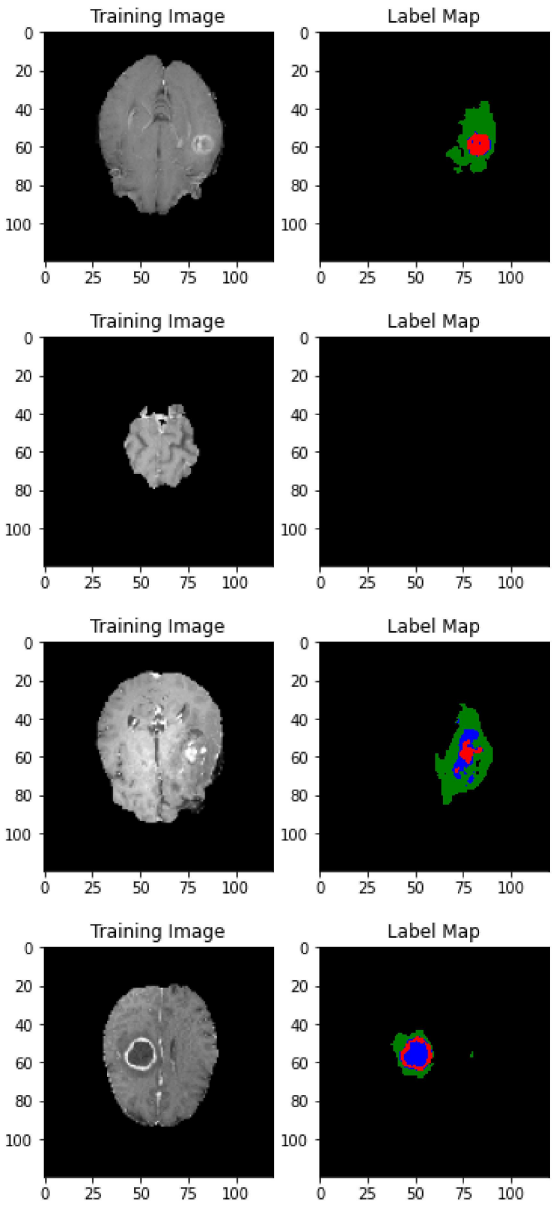
Suggested colour map for brain MR image:

```
cmap = 'gray'
```

Suggested colour map for segmentation map:

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

In [ ]:
```
### Insert your code ###
#initialize image and label paths
image_path = '/content/Task01_BrainTumour_2D/training_images'
label_path = '/content/Task01_BrainTumour_2D/training_labels'

#randomly shuffle all images
random_images = random.sample(os.listdir(image_path),4)

for image_names in random_images:
  image = imageio.imread(os.path.join(image_path, image_names))

  label = imageio.imread(os.path.join(label_path, image_names))

  fig, axs = plt.subplots(1, 2)
  axs[0].imshow(image, cmap = 'gray')
  axs[0].set_title('Training Image')
  axs[1].imshow(label, cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
  axs[1].set_title('Label Map')
### 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.

In [ ]:
```python
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.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.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 = [], []
        image_number = range(self.__len__())
        ### Insert your code ###

        # randomly choose a number of indices according to batch size
        indices = random.sample(image_number, batch_size)

        for i in indices:
          image, label = self.__getitem__(i)
          images.append(image)
          labels.append(label)

        images = np.expand_dims(images, axis = 1)
        # convert lists to arrays
        images = np.array(images)
        labels = 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.

In [ ]:
```
""" 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 estimates
        # of its computed mean and variance, which are then used for normalization
        # 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 ###

        # 128x64 64x64
        self.transpose3 = nn.ConvTranspose2d(n, int(n / 2), kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv3 = nn.Sequential(
            nn.Conv2d(n, int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU(),
            nn.Conv2d(int(n / 2), int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU()
        )

        # 64x32 32x32
        n = int(n / 2)
        self.transpose2 = nn.ConvTranspose2d(n, int(n / 2), kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv2 = nn.Sequential(
            nn.Conv2d(n, int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU(),
            nn.Conv2d(int(n / 2), int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU()
        )

        # 32x16 16x16
        n = int(n / 2)
        self.transpose1 = nn.ConvTranspose2d(n, int(n / 2), kernel_size=3, stride=2, padding=1, output_padding=1)
        self.deconv1 = nn.Sequential(
            nn.Conv2d(n, int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU(),
            nn.Conv2d(int(n / 2), int(n / 2), kernel_size=3, padding=1),
            nn.BatchNorm2d(int(n / 2)),
            nn.ReLU()
        )

        # # output
        self.deconv0 = nn.Conv2d(int(n / 2), 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.transpose3(x)
        x = torch.cat((conv3_skip, x), dim=1)
        x = self.deconv3(x)

        x = self.transpose2(x)
        x = torch.cat((conv2_skip, x), dim=1)
        x = self.deconv2(x)
```

```
        x = self.transpose1(x)
        x = torch.cat((conv1_skip, x), dim=1)
        x = self.deconv1(x)

        x = self.deconv0(x)

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

4. Train the segmentation model.

In [ ]:
```
# 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_BrainTumour_2D/training_labels')
test_set = BrainImageSet('Task01_BrainTumour_2D/test_images', 'Task01_BrainTumour_2D/test_labels')

# Train the model
# Note: when you debug the model, you may reduce the number of iterations or batch size to save time.
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 modules, e.g. dropout or batchnorm.
    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, dtype=torch.long)
    logits = model(images)

    # Perform optimisation and print out the training loss
    ### Insert your code ###
    optimizer.zero_grad()
    loss = criterion(logits, labels)
    loss.backward()
    optimizer.step()

    if it % 100 == 0:
        print('Iteration {0:5d}, loss = {1:.4f}'.format(it, loss.item()))
    ### End of your code ###

    # Evaluate
    if it % 100 == 0:
        model.eval()
        # Disabling gradient calculation during reference to reduce memory consumption
        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, test_labels = torch.from_numpy(test_images), torch.from_numpy(test_labels)
            test_images, test_labels = test_images.to(device, dtype=torch.float32), test_labels.to(device, dtype=torch.long)
            test_logits = model(test_images)
            test_loss = criterion(test_logits, test_labels)
            print('Test loss: {:.4f}'.format(test_loss.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'.format(it)))
print('Training took {:.3f}s in total.'.format(time.time() - start))
```

```
Device: cuda
Iteration   100, loss = 0.4109
Test loss: 0.4493
Iteration   200, loss = 0.1599
Test loss: 0.1920
Iteration   300, loss = 0.0858
Test loss: 0.1168
Iteration   400, loss = 0.0791
Test loss: 0.1062
Iteration   500, loss = 0.0688
Test loss: 0.0812
Iteration   600, loss = 0.0670
Test loss: 0.0821
Iteration   700, loss = 0.0552
Test loss: 0.0601
Iteration   800, loss = 0.0704
Test loss: 0.0661
Iteration   900, loss = 0.0400
Test loss: 0.0610
Iteration  1000, loss = 0.0579
Test loss: 0.0477
Iteration  1100, loss = 0.0322
Test loss: 0.0443
Iteration  1200, loss = 0.0337
Test loss: 0.0725
Iteration  1300, loss = 0.0231
Test loss: 0.0426
Iteration  1400, loss = 0.0297
Test loss: 0.0624
Iteration  1500, loss = 0.0347
Test loss: 0.0321
Iteration  1600, loss = 0.0299
Test loss: 0.0320
Iteration  1700, loss = 0.0394
Test loss: 0.0668
Iteration  1800, loss = 0.0441
Test loss: 0.0431
Iteration  1900, loss = 0.0439
Test loss: 0.0423
Iteration  2000, loss = 0.0460
Test loss: 0.0419
Iteration  2100, loss = 0.0251
Test loss: 0.0496
Iteration  2200, loss = 0.0412
Test loss: 0.0420
Iteration  2300, loss = 0.0332
Test loss: 0.0341
Iteration  2400, loss = 0.0196
Test loss: 0.0221
Iteration  2500, loss = 0.0490
Test loss: 0.0487
Iteration  2600, loss = 0.0249
Test loss: 0.0378
Iteration  2700, loss = 0.0223
Test loss: 0.0413
Iteration  2800, loss = 0.0346
Test loss: 0.0406
Iteration  2900, loss = 0.0209
Test loss: 0.0402
Iteration  3000, loss = 0.0236
Test loss: 0.0278
Iteration  3100, loss = 0.0205
Test loss: 0.0409
Iteration  3200, loss = 0.0255
Test loss: 0.0471
Iteration  3300, loss = 0.0260
Test loss: 0.0627
Iteration  3400, loss = 0.0256
Test loss: 0.0593
Iteration  3500, loss = 0.0154
Test loss: 0.0469
Iteration  3600, loss = 0.0293
Test loss: 0.0362
Iteration  3700, loss = 0.0192
Test loss: 0.0220
Iteration  3800, loss = 0.0205
Test loss: 0.0387
Iteration  3900, loss = 0.0218
Test loss: 0.0357
Iteration  4000, loss = 0.0299
Test loss: 0.0246
Iteration  4100, loss = 0.0124
Test loss: 0.0449
Iteration  4200, loss = 0.0148
Test loss: 0.0358
```

```
Iteration  4300, loss = 0.0180
Test loss: 0.0346
Iteration  4400, loss = 0.0238
Test loss: 0.0284
Iteration  4500, loss = 0.0181
Test loss: 0.0365
Iteration  4600, loss = 0.0198
Test loss: 0.0441
Iteration  4700, loss = 0.0165
Test loss: 0.0543
Iteration  4800, loss = 0.0201
Test loss: 0.0376
Iteration  4900, loss = 0.0146
Test loss: 0.0398
Iteration  5000, loss = 0.0203
Test loss: 0.0609
Iteration  5100, loss = 0.0166
Test loss: 0.0230
Iteration  5200, loss = 0.0140
Test loss: 0.0665
Iteration  5300, loss = 0.0116
Test loss: 0.0429
Iteration  5400, loss = 0.0186
Test loss: 0.0526
Iteration  5500, loss = 0.0178
Test loss: 0.0368
Iteration  5600, loss = 0.0150
Test loss: 0.0428
Iteration  5700, loss = 0.0109
Test loss: 0.0592
Iteration  5800, loss = 0.0142
Test loss: 0.0480
Iteration  5900, loss = 0.0119
Test loss: 0.0518
Iteration  6000, loss = 0.0156
Test loss: 0.0449
Iteration  6100, loss = 0.0176
Test loss: 0.0537
Iteration  6200, loss = 0.0148
Test loss: 0.0330
Iteration  6300, loss = 0.0101
Test loss: 0.0417
Iteration  6400, loss = 0.0147
Test loss: 0.0585
Iteration  6500, loss = 0.0101
Test loss: 0.0514
Iteration  6600, loss = 0.0149
Test loss: 0.0481
Iteration  6700, loss = 0.0196
Test loss: 0.0518
Iteration  6800, loss = 0.0129
Test loss: 0.0608
Iteration  6900, loss = 0.0123
Test loss: 0.0431
Iteration  7000, loss = 0.0108
Test loss: 0.0464
Iteration  7100, loss = 0.0185
Test loss: 0.0302
Iteration  7200, loss = 0.0083
Test loss: 0.0434
Iteration  7300, loss = 0.0127
Test loss: 0.0568
Iteration  7400, loss = 0.0181
Test loss: 0.0493
Iteration  7500, loss = 0.0107
Test loss: 0.0596
Iteration  7600, loss = 0.0088
Test loss: 0.0410
Iteration  7700, loss = 0.0125
Test loss: 0.0309
Iteration  7800, loss = 0.0135
Test loss: 0.0271
Iteration  7900, loss = 0.0093
Test loss: 0.0175
Iteration  8000, loss = 0.0176
Test loss: 0.0302
Iteration  8100, loss = 0.0125
Test loss: 0.0400
Iteration  8200, loss = 0.0080
Test loss: 0.0543
Iteration  8300, loss = 0.0110
Test loss: 0.0760
Iteration  8400, loss = 0.0138
Test loss: 0.0549
Iteration  8500, loss = 0.0141
```

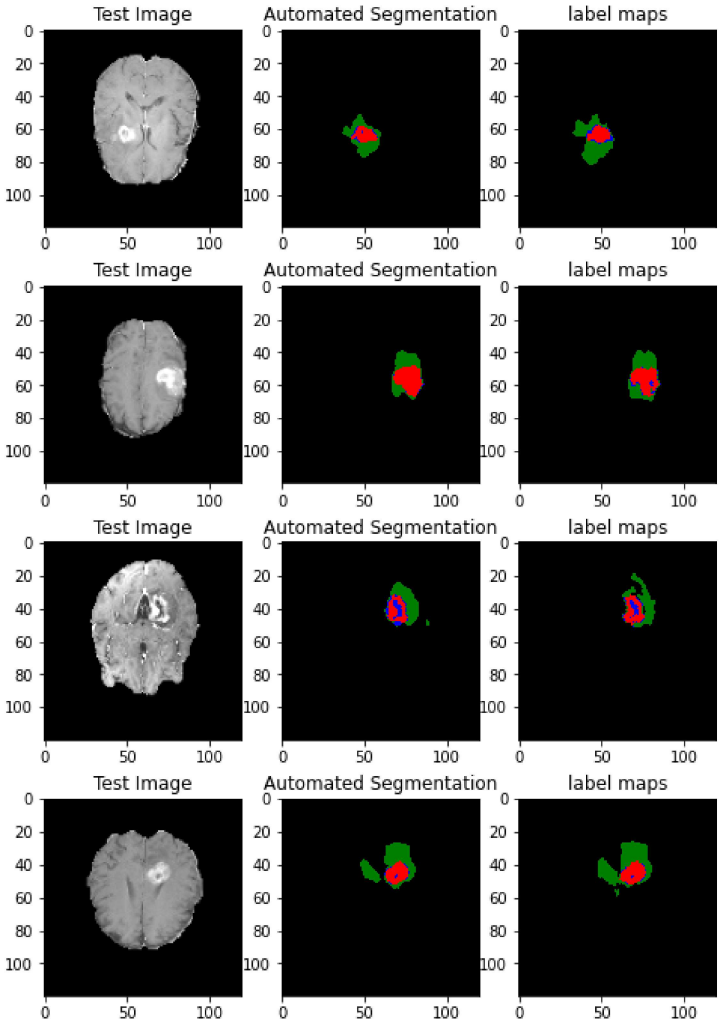
```
Test loss: 0.0331
Iteration  8600, loss = 0.0053
Test loss: 0.0641
Iteration  8700, loss = 0.0097
Test loss: 0.0601
Iteration  8800, loss = 0.0115
Test loss: 0.0548
Iteration  8900, loss = 0.0100
Test loss: 0.0501
Iteration  9000, loss = 0.0119
Test loss: 0.0504
Iteration  9100, loss = 0.0089
Test loss: 0.0421
Iteration  9200, loss = 0.0117
Test loss: 0.0302
Iteration  9300, loss = 0.0124
Test loss: 0.0472
Iteration  9400, loss = 0.0105
Test loss: 0.0587
Iteration  9500, loss = 0.0146
Test loss: 0.0524
Iteration  9600, loss = 0.0067
Test loss: 0.0879
Iteration  9700, loss = 0.0076
Test loss: 0.0627
Iteration  9800, loss = 0.0103
Test loss: 0.0500
Iteration  9900, loss = 0.0097
Test loss: 0.0480
Iteration 10000, loss = 0.0132
Test loss: 0.0414
Training took 372.915s 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.

In [ ]:

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

with torch.no_grad():
    test_images, test_labels = test_set.get_random_batch(4)
    test_images, test_labels = torch.from_numpy(test_images), torch.from_numpy(test_labels)
    test_images, test_labels = test_images.to(device, dtype=torch.float32), test_labels.to(device, dtype=torch.long)
    test_logits = model(test_images)

test_images_array = np.squeeze(torch.Tensor.cpu(test_images), axis=1)
test_logits_array = torch.argmax(test_logits, dim=1).cpu().numpy()
test_labels_array = np.squeeze(torch.Tensor.cpu(test_labels), axis=1)


# Plot the results
fig, axs = plt.subplots(4, 3, figsize=(8, 12))

for i in range(4):
    # Plot the test image
    axs[i, 0].imshow(test_images_array[i], cmap='gray')
    axs[i, 0].set_title('Test Image')

    # Plot the automated segmentation
    axs[i, 1].imshow(test_logits_array[i], cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
    axs[i, 1].set_title('Automated Segmentation')

    # Plot the ground truth segmentation
    axs[i, 2].imshow(test_labels_array[i], cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
    axs[i, 2].set_title('label maps')

plt.show()
### 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?

My trained model works relatively well with minimum training loss of 0.0053 and minimum test loss of 0.0175 for each batch and the automated segmentation map shows relatively accurate results compared with the ground truth label maps. However, there are several factors that require extra consideration before deploying this model to real clinic. Firstly, the model should be trained with more diverse and complex dataset to increase its generalization and avoid overfitting when encountered new data. This can be done by training witn more new data from other hospitals and sources or applying data augmentation to the existing dataset. For example, rotation, flip, zoom and shift. Secondly, more advanced optimization and regularization techniques could also be applied to reduce the problem of overfitting and improve generalization. Thirdly, the current groud truth label maps can only display segmentation graphs but lacks descriptions and annotations. Therefore, natural language processing models could be integrated to the existing model to provide more clinic-applicable functions that can generate descriptions to the segmentation map. Finally, ethics is another important aspect. Whether the new patient's brain image scan could be used to further train the model is controversial and should be handled with extra care.