coursework_02

February 14, 2025

1 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 ###
```

1.1 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.

1.2 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]`

1.3 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.

[1]:
```python
# 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
```

1.4 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

[2]:
```python
# 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()
```

```
--2025-02-14 10:22:26--
https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
Resolving www.dropbox.com (www.dropbox.com)… 162.125.64.18,
2620:100:6022:18::a27d:4212
Connecting to www.dropbox.com (www.dropbox.com)|162.125.64.18|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://www.dropbox.com/scl/fi/4bf8fqcfgf3lebiv2in99/Task01_BrainTumou
r_2D.tar.gz?rlkey=ceq898g2tr3aaxjxn4xjxbob1 [following]
--2025-02-14 10:22:26--  https://www.dropbox.com/scl/fi/4bf8fqcfgf3lebiv2in99/Ta
```

```
sk01_BrainTumour_2D.tar.gz?rlkey=ceq898g2tr3aaxjxn4xjxbob1
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response… 302 Found
Location: https://ucc9665a6e40db6d63b42af072ad.dl.dropboxusercontent.com/cd/0/in
line/CkGmMKmEeCj9-EFrec5q4ypZLI2WGTK0J12OHGO8rvUkuMPKGNFGLp1-2a9kpnzp1p7taiCFG3l
k6JBRNQ69LGo3psuP_XuAXuF7K2_4HViXUE_nh5mx7yEsN8vk5fhpH-4/file# [following]
--2025-02-14 10:22:27--  https://ucc9665a6e40db6d63b42af072ad.dl.dropboxusercont
ent.com/cd/0/inline/CkGmMKmEeCj9-EFrec5q4ypZLI2WGTK0J12OHGO8rvUkuMPKGNFGLp1-2a9k
pnzp1p7taiCFG3lk6JBRNQ69LGo3psuP_XuAXuF7K2_4HViXUE_nh5mx7yEsN8vk5fhpH-4/file
Resolving ucc9665a6e40db6d63b42af072ad.dl.dropboxusercontent.com
(ucc9665a6e40db6d63b42af072ad.dl.dropboxusercontent.com)… 162.125.64.15,
2620:100:6022:15::a27d:420f
Connecting to ucc9665a6e40db6d63b42af072ad.dl.dropboxusercontent.com
(ucc9665a6e40db6d63b42af072ad.dl.dropboxusercontent.com)|162.125.64.15|:443…
connected.
HTTP request sent, awaiting response… 302 Found
Location: /cd/0/inline2/CkHOGNUPBKgwmNCcTA0AK5JfOnncRzxXzoskJ7b8JC9KSq3LWDDBu1Pm
ML2OrayTDD3CqjJs6mK_6d5MIPwt9C39EmJJZvf5LqQxn5g4vGanf6J1wpJ83Xx38a8jlsTToLLqxrqc
ARzTPE0xdnTl1zfmbC3zbe7a7GrZNPmUePTY3h5no2GLetu4q4oUc9wOXruy2huYgU__Wa56QpV6ONyI
Z3ZUKV0M9E4LYsHIlN6M3c9r7bPAQFh7oT-y03Xzp_08cY13hmEbvuENadfRd0nKr5ZBsjMEKkvCkuVK
9G9UScCmnpI08klmK9nDOXccGKnAsnHUMs2x6ikHyZ1Cwwrp3a-SXbmnU2u5YyGQS5PI6Q/file
[following]
--2025-02-14 10:22:27--  https://ucc9665a6e40db6d63b42af072ad.dl.dropboxusercont
ent.com/cd/0/inline2/CkHOGNUPBKgwmNCcTA0AK5JfOnncRzxXzoskJ7b8JC9KSq3LWDDBu1PmML2
OrayTDD3CqjJs6mK_6d5MIPwt9C39EmJJZvf5LqQxn5g4vGanf6J1wpJ83Xx38a8jlsTToLLqxrqcARz
TPE0xdnTl1zfmbC3zbe7a7GrZNPmUePTY3h5no2GLetu4q4oUc9wOXruy2huYgU__Wa56QpV6ONyIZ3Z
UKV0M9E4LYsHIlN6M3c9r7bPAQFh7oT-y03Xzp_08cY13hmEbvuENadfRd0nKr5ZBsjMEKkvCkuVK9G9
UScCmnpI08klmK9nDOXccGKnAsnHUMs2x6ikHyZ1Cwwrp3a-SXbmnU2u5YyGQS5PI6Q/file
Reusing existing connection to
ucc9665a6e40db6d63b42af072ad.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’

Task01_BrainTumour_ 100%[===================>]   8.82M  7.19MB/s    in 1.2s

2025-02-14 10:22:29 (7.19 MB/s) - ‘Task01_BrainTumour_2D.tar.gz’ saved
[9251149/9251149]
```

1.5 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'])`

[3]:
```python
### Insert your code ###

# Visualise a random set of 4 training images along with their label maps
fig, axes = plt.subplots(4, 2, figsize=(10, 20))

# Suggested colour maps
image_cmap = 'gray'
label_cmap = colors.ListedColormap(['black', 'green', 'blue', 'red'])

# Load the images and labels
image_path = 'Task01_BrainTumour_2D/training_images'
label_path = 'Task01_BrainTumour_2D/training_labels'
image_names = sorted(os.listdir(image_path))
label_names = sorted(os.listdir(label_path))

# Randomly select 4 images
random_indices = random.sample(range(len(image_names)), 4)

for i, idx in enumerate(random_indices):
    image = imageio.imread(os.path.join(image_path, image_names[idx]))
    label = imageio.imread(os.path.join(label_path, label_names[idx]))

    axes[i, 0].imshow(image, cmap=image_cmap)
    axes[i, 0].set_title(f'Image {idx}')
    axes[i, 0].axis('off')

    axes[i, 1].imshow(label, cmap=label_cmap)
    axes[i, 1].set_title(f'Label {idx}')
    axes[i, 1].axis('off')

plt.tight_layout()
plt.show()


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

```
/tmp/ipykernel_23988/3903710960.py:20: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(os.path.join(image_path, image_names[idx]))
/tmp/ipykernel_23988/3903710960.py:21: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  label = imageio.imread(os.path.join(label_path, label_names[idx]))
```

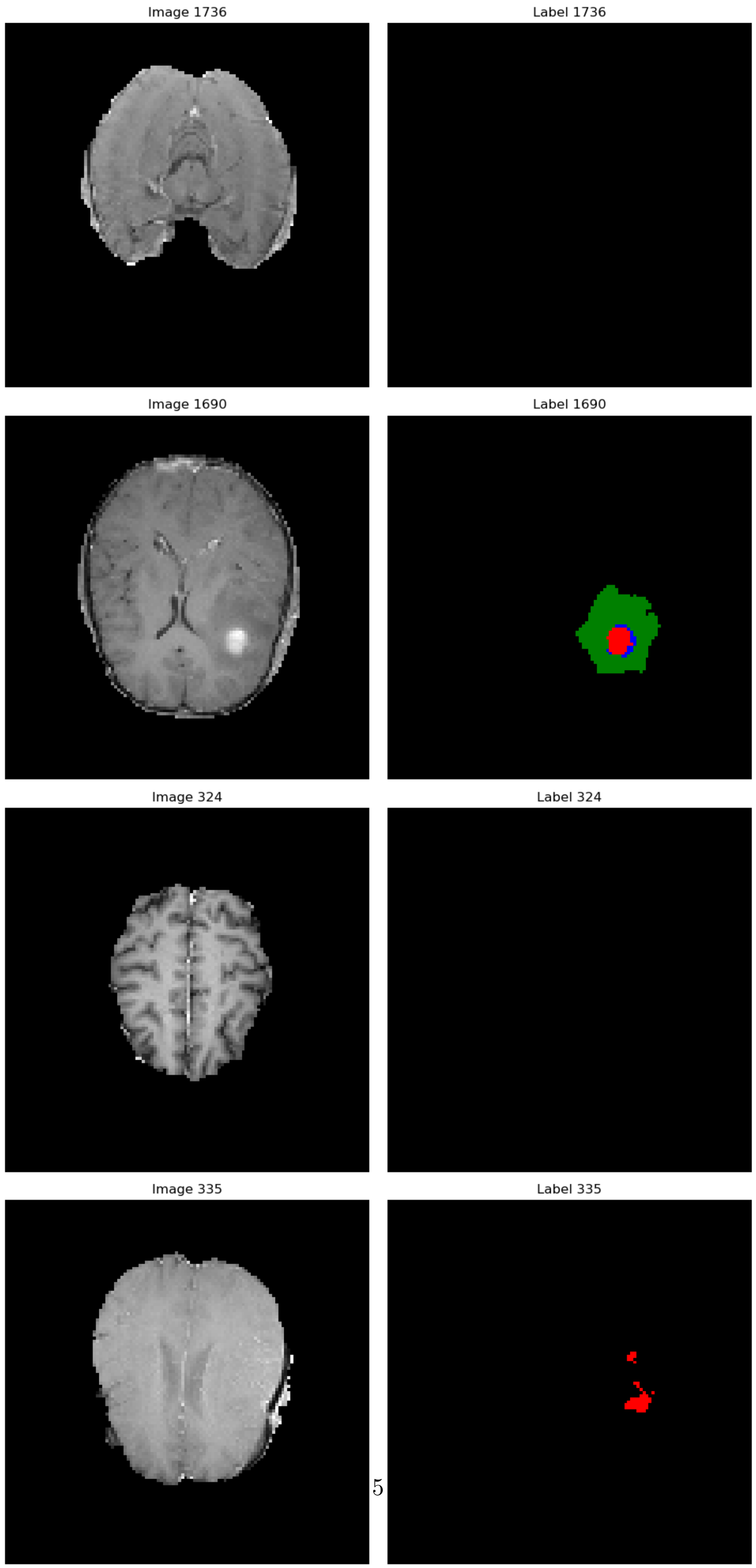


Image 1736
Label 1736
Image 1690
Label 1690
Image 324
Label 324
Image 335
Label 335

1.6 2. Implement a dataset class.

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

[4]:
```
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 = [], []

        ### Insert your code ###
        indices = random.sample(range(len(self.images)), batch_size)
        for idx in indices:
            image, label = self.__getitem__(idx)
            images.append(image)
            labels.append(label)

        images = np.expand_dims(np.array(images), axis=1)  # Add channel␣
↪dimension
        labels = np.array(labels)
        ### End of your code ###
        return images, labels
```

1.7 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.

[5]:
```
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()
        )

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

        self.conv3 = nn.Sequential(
            nn.Conv2d(2*n, 4*n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(4*n),
            nn.ReLU(),
            nn.Conv2d(4*n, 4*n, kernel_size=3, padding=1),
            nn.BatchNorm2d(4*n),
            nn.ReLU()
        )

        self.conv4 = nn.Sequential(
            nn.Conv2d(4*n, 8*n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(8*n),
            nn.ReLU(),
            nn.Conv2d(8*n, 8*n, kernel_size=3, padding=1),
            nn.BatchNorm2d(8*n),
            nn.ReLU()
        )

        # Decoder path
        self.upconv4 = nn.ConvTranspose2d(8*n, 4*n, kernel_size=2, stride=2)
        self.conv_up4 = nn.Sequential(
            nn.Conv2d(8*n, 4*n, kernel_size=3, padding=1),  # 8*n because of␣
↪concatenation
            nn.BatchNorm2d(4*n),
            nn.ReLU(),
            nn.Conv2d(4*n, 4*n, kernel_size=3, padding=1),
            nn.BatchNorm2d(4*n),
            nn.ReLU()
        )

        self.upconv3 = nn.ConvTranspose2d(4*n, 2*n, kernel_size=2, stride=2)
        self.conv_up3 = nn.Sequential(
            nn.Conv2d(4*n, 2*n, kernel_size=3, padding=1),  # 4*n because of␣
↪concatenation
            nn.BatchNorm2d(2*n),
            nn.ReLU(),
            nn.Conv2d(2*n, 2*n, kernel_size=3, padding=1),
            nn.BatchNorm2d(2*n),
            nn.ReLU()
        )

        self.upconv2 = nn.ConvTranspose2d(2*n, n, kernel_size=2, stride=2)
        self.conv_up2 = nn.Sequential(
            nn.Conv2d(2*n, n, kernel_size=3, padding=1),  # 2*n because of␣
↪concatenation
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

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

    def forward(self, x):
        # Encoder
        x1 = self.conv1(x)
        x2 = self.conv2(x1)
        x3 = self.conv3(x2)
        x4 = self.conv4(x3)

        # Decoder
        x = self.upconv4(x4)
        x = torch.cat((x, x3), dim=1)
        x = self.conv_up4(x)

        x = self.upconv3(x)
        x = torch.cat((x, x2), dim=1)
        x = self.conv_up3(x)

        x = self.upconv2(x)
        x = torch.cat((x, x1), dim=1)
        x = self.conv_up2(x)

        x = self.final_conv(x)

        return x
```

1.8 4. Train the segmentation model.

[6]:
```python
# 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()
    # print(f'Iteration {it}, Loss: {loss.item():.4f}, Time: {time.time() -␣
↪start_iter:.2f}s')
    ### End of your code ###

    # Evaluate
    if it % 100 == 0:
        print(f'Iteration {it}, Loss: {loss.item():.4f}, Time: {time.time() -␣
↪start_iter:.2f}s')
        # 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(f'Test Loss: {test_loss.item():.4f}')
            ### 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
```

```
/tmp/ipykernel_23988/3272815305.py:23: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(os.path.join(image_path, image_name))
/tmp/ipykernel_23988/3272815305.py:29: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
```

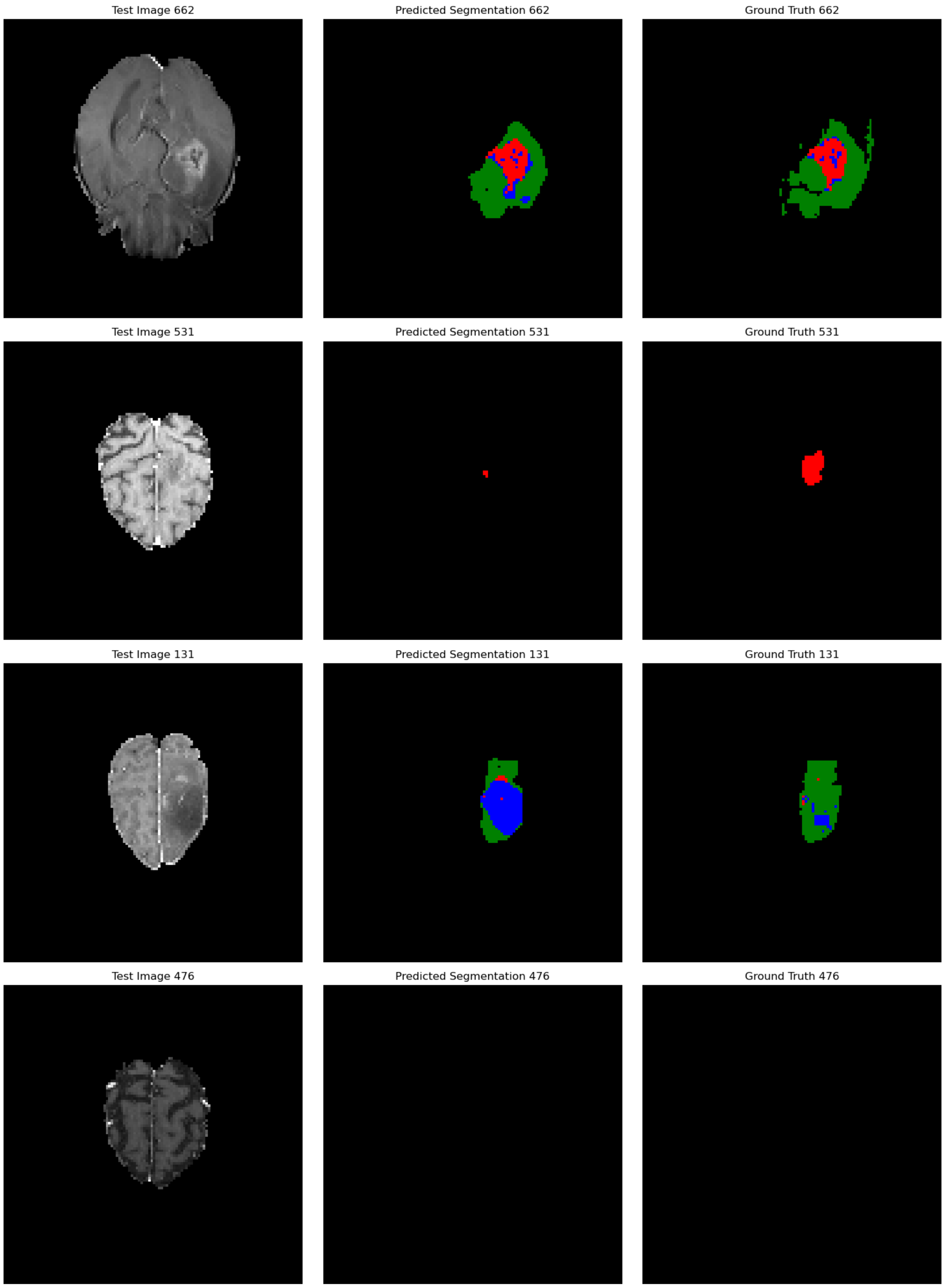
```
  label = imageio.imread(label_name)
Iteration 100, Loss: 0.5212, Time: 0.03s
Iteration 200, Loss: 0.2106, Time: 0.04s
Iteration 300, Loss: 0.1130, Time: 0.03s
Iteration 400, Loss: 0.0943, Time: 0.03s
Iteration 500, Loss: 0.0791, Time: 0.04s
Iteration 600, Loss: 0.0718, Time: 0.03s
Iteration 700, Loss: 0.0642, Time: 0.03s
Iteration 800, Loss: 0.0525, Time: 0.03s
Iteration 900, Loss: 0.0463, Time: 0.04s
Iteration 1000, Loss: 0.0529, Time: 0.03s
Iteration 1100, Loss: 0.0649, Time: 0.03s
Iteration 1200, Loss: 0.0444, Time: 0.03s
Iteration 1300, Loss: 0.0308, Time: 0.03s
Iteration 1400, Loss: 0.0350, Time: 0.03s
Iteration 1500, Loss: 0.0437, Time: 0.03s
Iteration 1600, Loss: 0.0417, Time: 0.03s
Iteration 1700, Loss: 0.0457, Time: 0.04s
Iteration 1800, Loss: 0.0436, Time: 0.04s
Iteration 1900, Loss: 0.0446, Time: 0.03s
Iteration 2000, Loss: 0.0227, Time: 0.03s
Iteration 2100, Loss: 0.0299, Time: 0.04s
Iteration 2200, Loss: 0.0461, Time: 0.03s
Iteration 2300, Loss: 0.0309, Time: 0.04s
Iteration 2400, Loss: 0.0336, Time: 0.04s
Iteration 2500, Loss: 0.0260, Time: 0.04s
Iteration 2600, Loss: 0.0195, Time: 0.04s
Iteration 2700, Loss: 0.0165, Time: 0.03s
Iteration 2800, Loss: 0.0235, Time: 0.04s
Iteration 2900, Loss: 0.0221, Time: 0.04s
Iteration 3000, Loss: 0.0322, Time: 0.04s
Iteration 3100, Loss: 0.0149, Time: 0.03s
Iteration 3200, Loss: 0.0274, Time: 0.04s
Iteration 3300, Loss: 0.0182, Time: 0.03s
Iteration 3400, Loss: 0.0273, Time: 0.03s
Iteration 3500, Loss: 0.0181, Time: 0.04s
Iteration 3600, Loss: 0.0243, Time: 0.04s
Iteration 3700, Loss: 0.0159, Time: 0.03s
Iteration 3800, Loss: 0.0139, Time: 0.04s
Iteration 3900, Loss: 0.0154, Time: 0.04s
Iteration 4000, Loss: 0.0189, Time: 0.04s
Iteration 4100, Loss: 0.0187, Time: 0.04s
Iteration 4200, Loss: 0.0160, Time: 0.04s
Iteration 4300, Loss: 0.0168, Time: 0.03s
Iteration 4400, Loss: 0.0135, Time: 0.04s
Iteration 4500, Loss: 0.0126, Time: 0.04s
Iteration 4600, Loss: 0.0193, Time: 0.04s
Iteration 4700, Loss: 0.0196, Time: 0.04s
Iteration 4800, Loss: 0.0166, Time: 0.04s
Iteration 4900, Loss: 0.0091, Time: 0.04s
Iteration 5000, Loss: 0.0121, Time: 0.03s
Iteration 5100, Loss: 0.0109, Time: 0.04s
Iteration 5200, Loss: 0.0109, Time: 0.03s
Iteration 5300, Loss: 0.0170, Time: 0.03s
Iteration 5400, Loss: 0.0150, Time: 0.04s
Iteration 5500, Loss: 0.0186, Time: 0.04s
Iteration 5600, Loss: 0.0105, Time: 0.04s
Iteration 5700, Loss: 0.0134, Time: 0.03s
Iteration 5800, Loss: 0.0154, Time: 0.04s
Iteration 5900, Loss: 0.0105, Time: 0.03s
Iteration 6000, Loss: 0.0172, Time: 0.04s
Iteration 6100, Loss: 0.0145, Time: 0.04s
Iteration 6200, Loss: 0.0092, Time: 0.03s
Iteration 6300, Loss: 0.0101, Time: 0.04s
Iteration 6400, Loss: 0.0107, Time: 0.03s
Iteration 6500, Loss: 0.0107, Time: 0.04s
Iteration 6600, Loss: 0.0129, Time: 0.03s
Iteration 6700, Loss: 0.0065, Time: 0.03s
Iteration 6800, Loss: 0.0082, Time: 0.03s
Iteration 6900, Loss: 0.0105, Time: 0.03s
Iteration 7000, Loss: 0.0084, Time: 0.03s
Iteration 7100, Loss: 0.0091, Time: 0.04s
Iteration 7200, Loss: 0.0114, Time: 0.03s
Iteration 7300, Loss: 0.0080, Time: 0.03s
Iteration 7400, Loss: 0.0123, Time: 0.03s
Iteration 7500, Loss: 0.0088, Time: 0.03s
Iteration 7600, Loss: 0.0153, Time: 0.04s
Iteration 7700, Loss: 0.0086, Time: 0.03s
Iteration 7800, Loss: 0.0098, Time: 0.04s
Iteration 7900, Loss: 0.0164, Time: 0.04s
Iteration 8000, Loss: 0.0116, Time: 0.04s
Iteration 8100, Loss: 0.0108, Time: 0.04s
Iteration 8200, Loss: 0.0136, Time: 0.04s
Iteration 8300, Loss: 0.0124, Time: 0.04s
Iteration 8400, Loss: 0.0099, Time: 0.03s
Iteration 8500, Loss: 0.0143, Time: 0.03s
Iteration 8600, Loss: 0.0121, Time: 0.03s
Iteration 8700, Loss: 0.0137, Time: 0.04s
Iteration 8800, Loss: 0.0144, Time: 0.03s
Iteration 8900, Loss: 0.0095, Time: 0.04s
Iteration 9000, Loss: 0.0103, Time: 0.04s
Iteration 9100, Loss: 0.0106, Time: 0.03s
Iteration 9200, Loss: 0.0080, Time: 0.03s
Iteration 9300, Loss: 0.0085, Time: 0.03s
Iteration 9400, Loss: 0.0123, Time: 0.03s
Iteration 9500, Loss: 0.0103, Time: 0.03s
Iteration 9600, Loss: 0.0070, Time: 0.04s
Iteration 9700, Loss: 0.0070, Time: 0.03s
Iteration 9800, Loss: 0.0060, Time: 0.03s
Iteration 9900, Loss: 0.0163, Time: 0.04s
Iteration 10000, Loss: 0.0069, Time: 0.04s
Training took 241.938s in total.
```

1.9 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.

[7]:
```python
### Insert your code ###
# Visualise the segmentation results
# Load the test images and labels
test_image_path = 'Task01_BrainTumour_2D/test_images'
test_label_path = 'Task01_BrainTumour_2D/test_labels'
test_image_names = sorted(os.listdir(test_image_path))
test_label_names = sorted(os.listdir(test_label_path))

# Randomly select 4 test images
random_test_indices = random.sample(range(len(test_image_names)), 4)

fig, axes = plt.subplots(4, 3, figsize=(15, 20))

for i, idx in enumerate(random_test_indices):
    test_image = imageio.imread(os.path.join(test_image_path,␣
 ↪test_image_names[idx]))
    test_label = imageio.imread(os.path.join(test_label_path,␣
 ↪test_label_names[idx]))

    # Preprocess the test image
    test_image_norm = normalise_intensity(test_image)
    test_image_tensor = torch.from_numpy(test_image_norm).unsqueeze(0).
 ↪unsqueeze(0).to(device, dtype=torch.float32)

    # Get the model prediction
    model.eval()
    with torch.no_grad():
        pred = model(test_image_tensor)
        pred = torch.argmax(pred, dim=1).squeeze().cpu().numpy()

    # Plot the test image
    axes[i, 0].imshow(test_image, cmap=image_cmap)
    axes[i, 0].set_title(f'Test Image {idx}')
    axes[i, 0].axis('off')

    # Plot the predicted segmentation
    axes[i, 1].imshow(pred, cmap=label_cmap)
    axes[i, 1].set_title(f'Predicted Segmentation {idx}')
    axes[i, 1].axis('off')

    # Plot the ground truth segmentation
    axes[i, 2].imshow(test_label, cmap=label_cmap)
    axes[i, 2].set_title(f'Ground Truth {idx}')
    axes[i, 2].axis('off')

plt.tight_layout()
plt.show()
### End of your code ###
```

```
/tmp/ipykernel_23988/2287680980.py:15: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  test_image = imageio.imread(os.path.join(test_image_path,
test_image_names[idx]))
/tmp/ipykernel_23988/2287680980.py:16: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To keep
the current behavior (and make this warning disappear) use `import imageio.v2 as
imageio` or call `imageio.v2.imread` directly.
  test_label = imageio.imread(os.path.join(test_label_path,
test_label_names[idx]))
```


Test Image 662
Predicted Segmentation 662
Ground Truth 662
Test Image 531
Predicted Segmentation 531
Ground Truth 531
Test Image 131
Predicted Segmentation 131
Ground Truth 131
Test Image 476
Predicted Segmentation 476
Ground Truth 476

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

Yes, the trained model shows promising results in segmenting brain tumors from MRI images. The model's performance, as indicated by the training and test losses, suggests that it has learned to identify and segment different tumor regions effectively. However, there is still room for improvement to enhance its accuracy and robustness for clinical deployment.

Possible improvements:

1. Data Augmentation: Increase the diversity of the training data by applying data augmentation techniques such as rotation, scaling, flipping, and adding noise. This can help the model generalize better to unseen data.

2. Hyperparameter Tuning: Experiment with different hyperparameters such as learning rate, batch size, and the number of filters in each convolutional layer to find the optimal configuration for the model.

3. Model Architecture: Explore more advanced architectures such as U-Net++ or Attention U-Net, which have shown improved performance in medical image segmentation tasks.

4. Cross-validation: Use cross-validation to ensure that the model's performance is consistent across different subsets of the data and to avoid overfitting.

5. Clinical Validation: Collaborate with medical professionals to validate the model's performance on real clinical data and gather feedback for further improvements.