coursework_02

March 3, 2023

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.

[ ]:
```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

[ ]:
```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()
```

```
--2023-03-03 17:39:52--
https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
Resolving www.dropbox.com (www.dropbox.com)… 162.125.3.18,
2620:100:6018:18::a27d:312
Connecting to www.dropbox.com (www.dropbox.com)|162.125.3.18|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz [following]
--2023-03-03 17:39:52--
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://uc2105f412a554ed947969cd838a.dl.dropboxusercontent.com/cd/0/in
line/B3i3v2Ge0zbmefj1eGff7MVq44msrP-ZByTqE_XqVCP5bLLl6_qmX-ylOm1p0op9IOB5D1eNCmb
Zd6CRv3GWqZe6Tkyzg4o36qfelQ4dHB1Lqx8STWYgf0-1DeQlcnnaglf7fXLNPn-
9wLyNVnXPh5eln1RXJHK6UseizlktQY4bwQ/file# [following]
--2023-03-03 17:39:52--  https://uc2105f412a554ed947969cd838a.dl.dropboxusercont
ent.com/cd/0/inline/B3i3v2Ge0zbmefj1eGff7MVq44msrP-ZByTqE_XqVCP5bLLl6_qmX-ylOm1p
0op9IOB5D1eNCmbZd6CRv3GWqZe6Tkyzg4o36qfelQ4dHB1Lqx8STWYgf0-1DeQlcnnaglf7fXLNPn-
9wLyNVnXPh5eln1RXJHK6UseizlktQY4bwQ/file
Resolving uc2105f412a554ed947969cd838a.dl.dropboxusercontent.com
(uc2105f412a554ed947969cd838a.dl.dropboxusercontent.com)… 162.125.3.15,
2620:100:601b:15::a27d:80f
Connecting to uc2105f412a554ed947969cd838a.dl.dropboxusercontent.com
(uc2105f412a554ed947969cd838a.dl.dropboxusercontent.com)|162.125.3.15|:443…
connected.
HTTP request sent, awaiting response… 302 Found
Location: /cd/0/inline2/B3jQVB8mD_X0ExsiWp4yGrTFN93Yv9_6Er6fzJq1llYmiT02J0lNeCbI
uX08HbYaYSExEH7FfS1HlxohdAaJ8uLAs0Xock_6K9mDk1mIqjbvx8zppznhpAuMRj9_0avM2bTJPgnT
zxI2dA-FIE84986QELe1a-VDaLfNbM8_xTNijpwk_LVBHlSMi550Ai7R8u9CfvQXGTGpAfFcQG-
oPL2KDHXsmb92uHOBfp8EeR6rfRVcpPd5uhaMT4B-_O09F8BhL-
WAzZ_pq1St2EC_F_JPsYdzt16pqb1SL4YgNrPhxNskVL7JBQ7WlSECma3-xYB8KIZsIgkq-
Rv1ob5csLOCd07BMYqtdu0fG-
bKsC37ZjYTGfW88kycSTDOedP5pfKsIHqbQWdGXb0JSmdwhLQONkavxnuMTSAYrF2kJhEsjw/file
[following]
--2023-03-03 17:39:53--  https://uc2105f412a554ed947969cd838a.dl.dropboxusercont
ent.com/cd/0/inline2/B3jQVB8mD_X0ExsiWp4yGrTFN93Yv9_6Er6fzJq1llYmiT02J0lNeCbIuX0
8HbYaYSExEH7FfS1HlxohdAaJ8uLAs0Xock_6K9mDk1mIqjbvx8zppznhpAuMRj9_0avM2bTJPgnTzxI
2dA-FIE84986QELe1a-VDaLfNbM8_xTNijpwk_LVBHlSMi550Ai7R8u9CfvQXGTGpAfFcQG-
oPL2KDHXsmb92uHOBfp8EeR6rfRVcpPd5uhaMT4B-_O09F8BhL-
WAzZ_pq1St2EC_F_JPsYdzt16pqb1SL4YgNrPhxNskVL7JBQ7WlSECma3-xYB8KIZsIgkq-
Rv1ob5csLOCd07BMYqtdu0fG-
bKsC37ZjYTGfW88kycSTDOedP5pfKsIHqbQWdGXb0JSmdwhLQONkavxnuMTSAYrF2kJhEsjw/file
Reusing existing connection to
uc2105f412a554ed947969cd838a.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  55.5MB/s    in 0.2s

2023-03-03 17:39:53 (55.5 MB/s) - ‘Task01_BrainTumour_2D.tar.gz.1’ 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:

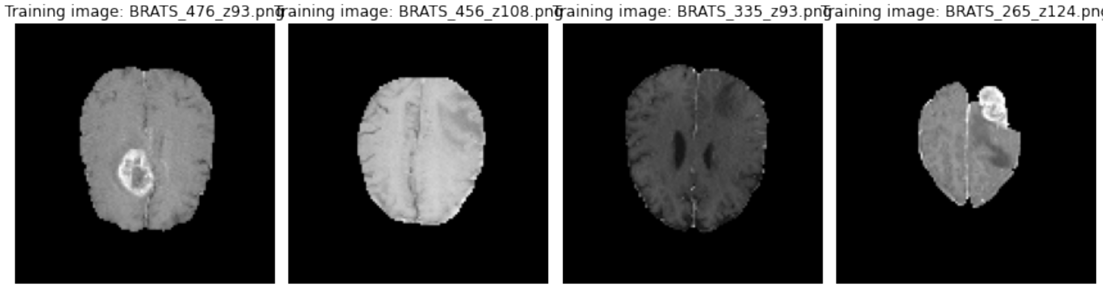
```
cmap = 'gray'
```

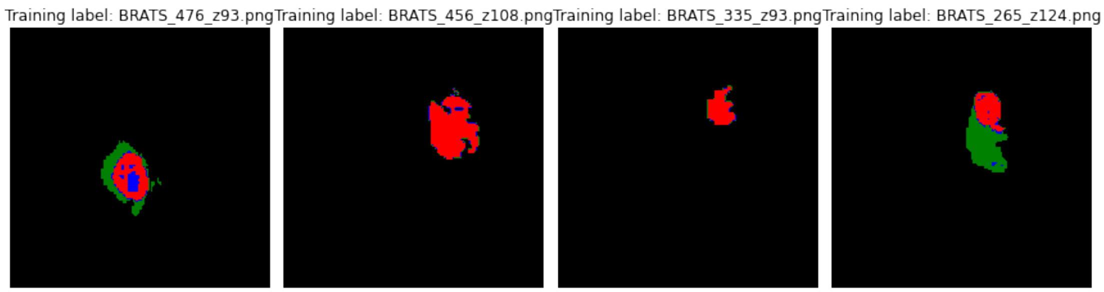
Suggested colour map for segmentation map:

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

[ ]:

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

n = 4
seg_cmap = colors.ListedColormap(['black', 'green', 'blue', 'red'])

random_set = [random.choice(os.listdir('Task01_BrainTumour_2D/training_images/
↪')) for i in range(n)]

for i in range(n):
    plt.subplot(2, n, i+1)
    plt.imshow(imageio.imread('Task01_BrainTumour_2D/training_images/' +␣
↪random_set[i]), cmap='gray')
    plt.axis('off')
    plt.title(f"Training image: {random_set[i]}")

    plt.subplot(2, n, i+n+1)
    plt.imshow(imageio.imread('Task01_BrainTumour_2D/training_labels/' +␣
↪random_set[i]), cmap=seg_cmap)
    plt.axis('off')
    plt.title(f"Training label: {random_set[i]}")

    plt.gcf().set_size_inches(12, 12)
    plt.tight_layout()

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





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.

[ ]:
```
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))
```

```python
        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 = [], []

        indices = random.sample(range(len(self.images)), batch_size)

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

        images = np.array(images)
        images = np.expand_dims(images, axis=1)
        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.

```
[ ]: """ 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 ###

        n = int(n / 2)  # 64
        self.conv5_up = nn.Sequential(
            nn.ConvTranspose2d(n * 2, n, kernel_size=3, stride=2, padding=1,
↪output_padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
        )

        self.conv5 = 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 = int(n / 2)  # 32
        self.conv6_up = nn.Sequential(
            nn.ConvTranspose2d(n * 2, n, kernel_size=3, stride=2, padding=1,
↪output_padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
        )

        self.conv6 = 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 = int(n / 2)  # 16
        self.conv7_up = nn.Sequential(
            nn.ConvTranspose2d(n * 2, n, kernel_size=3, stride=2, padding=1,␣
↪output_padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
        )

        self.conv7 = 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.conv8 = nn.Conv2d(n, output_channel, kernel_size=3, padding=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.conv5_up(x)
        x = torch.cat((x, conv3_skip), dim=1)
        x = self.conv5(x)

        x = self.conv6_up(x)
        x = torch.cat((x, conv2_skip), dim=1)
```

```
        x = self.conv6(x)

        x = self.conv7_up(x)
        x = torch.cat((x, conv1_skip), dim=1)
        x = self.conv7(x)

        x = self.conv8(x)

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

1.8 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_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):
```

```python
    # 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 ###

    loss = criterion(logits, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if it % 100 == 0:
        print(f'Iteration {it}: train loss = {loss:.3f}, time = {time.time() - start_iter:.3f}s')

    ### 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 ###

            images, labels = test_set.get_random_batch(eval_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)

            loss = criterion(logits, labels)
            print(f'Iteration {it}: test loss = {loss:.3f}')

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

```python
    # 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: train loss = 0.070, time = 0.055s
Iteration 100: test loss = 0.069
Iteration 200: train loss = 0.068, time = 0.056s
Iteration 200: test loss = 0.079
Iteration 300: train loss = 0.065, time = 0.050s
Iteration 300: test loss = 0.040
Iteration 400: train loss = 0.050, time = 0.057s
Iteration 400: test loss = 0.056
Iteration 500: train loss = 0.061, time = 0.057s
Iteration 500: test loss = 0.068
Iteration 600: train loss = 0.050, time = 0.052s
Iteration 600: test loss = 0.065
Iteration 700: train loss = 0.042, time = 0.053s
Iteration 700: test loss = 0.078
Iteration 800: train loss = 0.036, time = 0.059s
Iteration 800: test loss = 0.075
Iteration 900: train loss = 0.048, time = 0.057s
Iteration 900: test loss = 0.073
Iteration 1000: train loss = 0.029, time = 0.056s
Iteration 1000: test loss = 0.054
Iteration 1100: train loss = 0.045, time = 0.058s
Iteration 1100: test loss = 0.039
Iteration 1200: train loss = 0.054, time = 0.058s
Iteration 1200: test loss = 0.098
Iteration 1300: train loss = 0.048, time = 0.060s
Iteration 1300: test loss = 0.057
Iteration 1400: train loss = 0.030, time = 0.058s
Iteration 1400: test loss = 0.040
Iteration 1500: train loss = 0.033, time = 0.057s
Iteration 1500: test loss = 0.077
Iteration 1600: train loss = 0.061, time = 0.060s
Iteration 1600: test loss = 0.046
Iteration 1700: train loss = 0.036, time = 0.059s
Iteration 1700: test loss = 0.033
Iteration 1800: train loss = 0.025, time = 0.060s
Iteration 1800: test loss = 0.047
Iteration 1900: train loss = 0.031, time = 0.053s
Iteration 1900: test loss = 0.021
Iteration 2000: train loss = 0.055, time = 0.060s
Iteration 2000: test loss = 0.042
Iteration 2100: train loss = 0.038, time = 0.059s
```

```
Iteration 2100: test loss = 0.050
Iteration 2200: train loss = 0.024, time = 0.051s
Iteration 2200: test loss = 0.041
Iteration 2300: train loss = 0.022, time = 0.060s
Iteration 2300: test loss = 0.057
Iteration 2400: train loss = 0.030, time = 0.055s
Iteration 2400: test loss = 0.038
Iteration 2500: train loss = 0.028, time = 0.055s
Iteration 2500: test loss = 0.050
Iteration 2600: train loss = 0.025, time = 0.060s
Iteration 2600: test loss = 0.061
Iteration 2700: train loss = 0.014, time = 0.059s
Iteration 2700: test loss = 0.030
Iteration 2800: train loss = 0.024, time = 0.062s
Iteration 2800: test loss = 0.057
Iteration 2900: train loss = 0.020, time = 0.057s
Iteration 2900: test loss = 0.041
Iteration 3000: train loss = 0.018, time = 0.060s
Iteration 3000: test loss = 0.026
Iteration 3100: train loss = 0.034, time = 0.062s
Iteration 3100: test loss = 0.028
Iteration 3200: train loss = 0.015, time = 0.059s
Iteration 3200: test loss = 0.054
Iteration 3300: train loss = 0.023, time = 0.059s
Iteration 3300: test loss = 0.034
Iteration 3400: train loss = 0.023, time = 0.057s
Iteration 3400: test loss = 0.035
Iteration 3500: train loss = 0.017, time = 0.060s
Iteration 3500: test loss = 0.044
Iteration 3600: train loss = 0.022, time = 0.063s
Iteration 3600: test loss = 0.041
Iteration 3700: train loss = 0.010, time = 0.061s
Iteration 3700: test loss = 0.044
Iteration 3800: train loss = 0.027, time = 0.059s
Iteration 3800: test loss = 0.039
Iteration 3900: train loss = 0.021, time = 0.061s
Iteration 3900: test loss = 0.036
Iteration 4000: train loss = 0.017, time = 0.059s
Iteration 4000: test loss = 0.025
Iteration 4100: train loss = 0.023, time = 0.057s
Iteration 4100: test loss = 0.032
Iteration 4200: train loss = 0.020, time = 0.058s
Iteration 4200: test loss = 0.028
Iteration 4300: train loss = 0.014, time = 0.061s
Iteration 4300: test loss = 0.023
Iteration 4400: train loss = 0.021, time = 0.055s
Iteration 4400: test loss = 0.068
Iteration 4500: train loss = 0.013, time = 0.063s
```

```
Iteration 4500: test loss = 0.029
Iteration 4600: train loss = 0.010, time = 0.064s
Iteration 4600: test loss = 0.043
Iteration 4700: train loss = 0.009, time = 0.054s
Iteration 4700: test loss = 0.027
Iteration 4800: train loss = 0.017, time = 0.062s
Iteration 4800: test loss = 0.021
Iteration 4900: train loss = 0.016, time = 0.062s
Iteration 4900: test loss = 0.035
Iteration 5000: train loss = 0.019, time = 0.058s
Iteration 5000: test loss = 0.034
Iteration 5100: train loss = 0.013, time = 0.065s
Iteration 5100: test loss = 0.043
Iteration 5200: train loss = 0.013, time = 0.063s
Iteration 5200: test loss = 0.033
Iteration 5300: train loss = 0.013, time = 0.058s
Iteration 5300: test loss = 0.019
Iteration 5400: train loss = 0.008, time = 0.063s
Iteration 5400: test loss = 0.055
Iteration 5500: train loss = 0.022, time = 0.062s
Iteration 5500: test loss = 0.033
Iteration 5600: train loss = 0.016, time = 0.055s
Iteration 5600: test loss = 0.027
Iteration 5700: train loss = 0.022, time = 0.060s
Iteration 5700: test loss = 0.042
Iteration 5800: train loss = 0.006, time = 0.062s
Iteration 5800: test loss = 0.034
Iteration 5900: train loss = 0.016, time = 0.053s
Iteration 5900: test loss = 0.030
Iteration 6000: train loss = 0.012, time = 0.062s
Iteration 6000: test loss = 0.061
Iteration 6100: train loss = 0.012, time = 0.062s
Iteration 6100: test loss = 0.045
Iteration 6200: train loss = 0.007, time = 0.062s
Iteration 6200: test loss = 0.038
Iteration 6300: train loss = 0.017, time = 0.059s
Iteration 6300: test loss = 0.023
Iteration 6400: train loss = 0.014, time = 0.064s
Iteration 6400: test loss = 0.043
Iteration 6500: train loss = 0.014, time = 0.062s
Iteration 6500: test loss = 0.041
Iteration 6600: train loss = 0.011, time = 0.063s
Iteration 6600: test loss = 0.032
Iteration 6700: train loss = 0.011, time = 0.062s
Iteration 6700: test loss = 0.020
Iteration 6800: train loss = 0.011, time = 0.059s
Iteration 6800: test loss = 0.041
Iteration 6900: train loss = 0.010, time = 0.060s
```

```
Iteration 6900: test loss = 0.037
Iteration 7000: train loss = 0.012, time = 0.061s
Iteration 7000: test loss = 0.051
Iteration 7100: train loss = 0.008, time = 0.063s
Iteration 7100: test loss = 0.065
Iteration 7200: train loss = 0.012, time = 0.061s
Iteration 7200: test loss = 0.046
Iteration 7300: train loss = 0.016, time = 0.064s
Iteration 7300: test loss = 0.040
Iteration 7400: train loss = 0.012, time = 0.062s
Iteration 7400: test loss = 0.051
Iteration 7500: train loss = 0.014, time = 0.063s
Iteration 7500: test loss = 0.064
Iteration 7600: train loss = 0.011, time = 0.063s
Iteration 7600: test loss = 0.037
Iteration 7700: train loss = 0.010, time = 0.064s
Iteration 7700: test loss = 0.034
Iteration 7800: train loss = 0.008, time = 0.064s
Iteration 7800: test loss = 0.044
Iteration 7900: train loss = 0.011, time = 0.064s
Iteration 7900: test loss = 0.055
Iteration 8000: train loss = 0.007, time = 0.062s
Iteration 8000: test loss = 0.038
Iteration 8100: train loss = 0.009, time = 0.064s
Iteration 8100: test loss = 0.023
Iteration 8200: train loss = 0.009, time = 0.061s
Iteration 8200: test loss = 0.060
Iteration 8300: train loss = 0.010, time = 0.062s
Iteration 8300: test loss = 0.057
Iteration 8400: train loss = 0.012, time = 0.061s
Iteration 8400: test loss = 0.058
Iteration 8500: train loss = 0.006, time = 0.064s
Iteration 8500: test loss = 0.026
Iteration 8600: train loss = 0.010, time = 0.063s
Iteration 8600: test loss = 0.050
Iteration 8700: train loss = 0.007, time = 0.056s
Iteration 8700: test loss = 0.037
Iteration 8800: train loss = 0.010, time = 0.064s
Iteration 8800: test loss = 0.069
Iteration 8900: train loss = 0.010, time = 0.065s
Iteration 8900: test loss = 0.029
Iteration 9000: train loss = 0.009, time = 0.058s
Iteration 9000: test loss = 0.050
Iteration 9100: train loss = 0.007, time = 0.062s
Iteration 9100: test loss = 0.051
Iteration 9200: train loss = 0.008, time = 0.062s
Iteration 9200: test loss = 0.058
Iteration 9300: train loss = 0.011, time = 0.059s
```

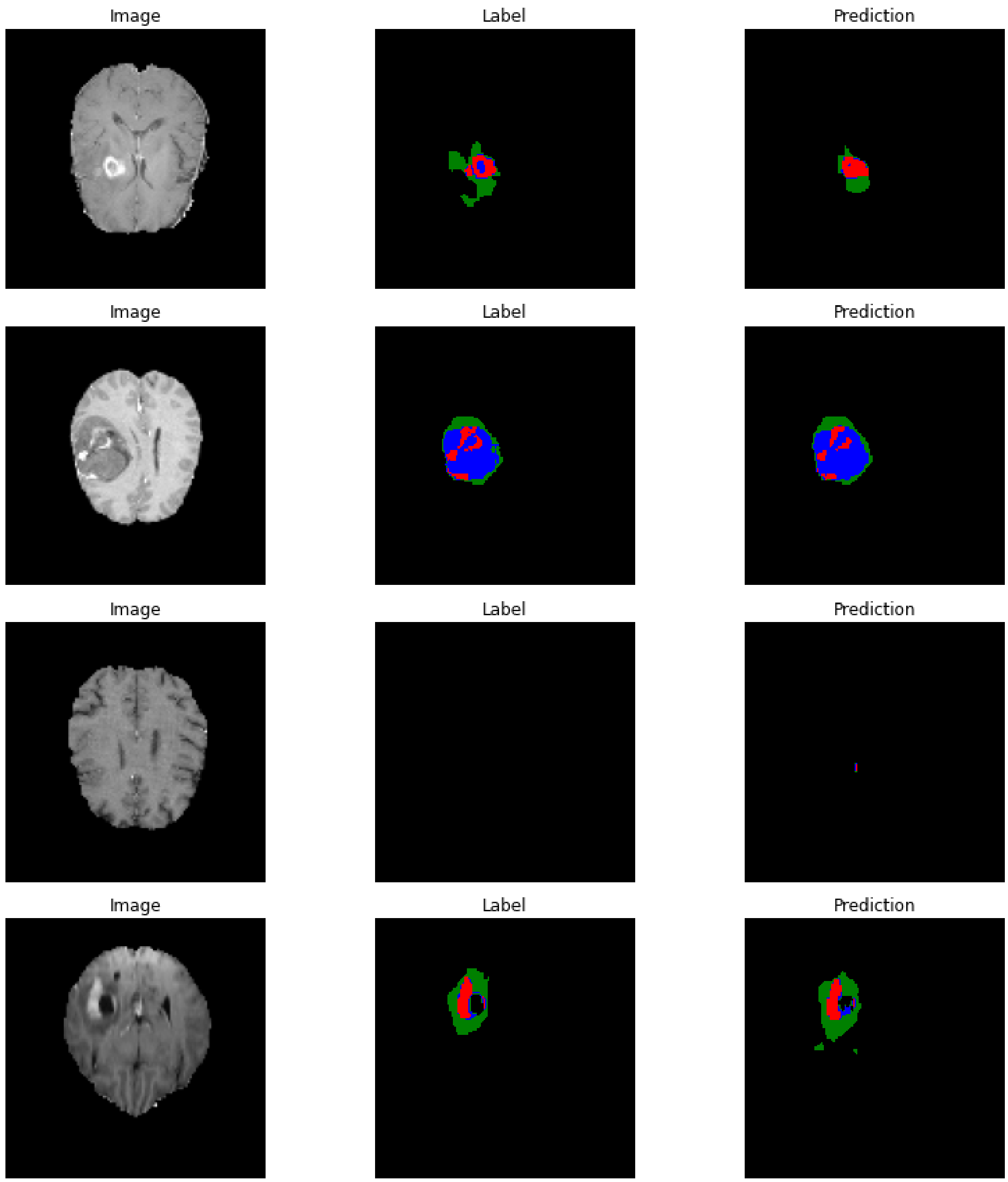
```
Iteration 9300: test loss = 0.026
Iteration 9400: train loss = 0.010, time = 0.062s
Iteration 9400: test loss = 0.046
Iteration 9500: train loss = 0.006, time = 0.061s
Iteration 9500: test loss = 0.033
Iteration 9600: train loss = 0.009, time = 0.056s
Iteration 9600: test loss = 0.072
Iteration 9700: train loss = 0.012, time = 0.062s
Iteration 9700: test loss = 0.025
Iteration 9800: train loss = 0.007, time = 0.062s
Iteration 9800: test loss = 0.053
Iteration 9900: train loss = 0.010, time = 0.055s
Iteration 9900: test loss = 0.045
Iteration 10000: train loss = 0.005, time = 0.063s
Iteration 10000: test loss = 0.056
Training took 400.743s 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.

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

model.load_state_dict(torch.load(os.path.join(model_dir, 'model_10000.pt')))
model.eval()

n = 4
images, labels = test_set.get_random_batch(n)
images, labels = torch.from_numpy(images), torch.from_numpy(labels)
images, labels = images.to(device, dtype=torch.float32), labels.to(device,␣
 ↪dtype=torch.long)

with torch.no_grad():
    logits = model(images)
    probs = torch.softmax(logits, dim=1)
    preds = torch.argmax(probs, dim=1).cpu().numpy()

fig, axs = plt.subplots(nrows=n, ncols=3, figsize=(12, 12))

for i in range(n):
    axs[i, 0].imshow(images[i, 0].cpu().numpy(), cmap='gray')
    axs[i, 0].set_title('Image')
    axs[i, 0].axis('off')

    axs[i, 1].imshow(labels[i].cpu().numpy(), cmap=seg_cmap)
    axs[i, 1].set_title('Label')
```

```
    axs[i, 1].axis('off')

    axs[i, 2].imshow(preds[i], cmap=seg_cmap)
    axs[i, 2].set_title('Prediction')
    axs[i, 2].axis('off')

plt.tight_layout()


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



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?

The model works fairly well with a final cross-entropy loss on the training set of 0.005 at the 10,000th iteration and a testing set loss of around 0.05. Also, as can be seen in task 5 by visualising some of the model's predicitons, we can see that the model identifies the presence of tumours quite accurately, although sometimes the colours (i.e. the tumour type) and/or tumour boundaries do not match.

Looking at a few examples is by no means a rigourous test however, and before deploying to a real life scenario we would need to compute other metrics such as f1 to get a better picture of the false positive/ false negative rate of the model. This is important as we would need to optimise the model for the clinic's needs e.g. maybe the clinic wants a low false negative rate so they do not miss any potential tumours, so we would try and improve the recall metric.

Furthermore, before deploying to a real clinic we could improve many hyperparameters of the model, such as batch size, learning rate, number/depth of layers etc., by using a grid search or more sophisticated methods.