# coursework_02

February 24, 2024

# 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.

[39]:
```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()
```

```
Will not apply HSTS. The HSTS database must be a regular and non-world-writable
file.
ERROR: could not open HSTS store at '/Users/fate/.wget-hsts'. HSTS will be
disabled.
--2024-02-19 12:58:48--
https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
Resolving www.dropbox.com (www.dropbox.com)… 2620:100:6020:18::a27d:4012,
162.125.64.18
Connecting to www.dropbox.com
```

```
(www.dropbox.com)|2620:100:6020:18::a27d:4012|:443… failed: Operation timed
out.
Connecting to www.dropbox.com (www.dropbox.com)|162.125.64.18|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz [following]
--2024-02-19 13:00:04--
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://uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com/cd/0/in
line/CNmmmw9wTypFuF6p7dk3UgcIXG7N-
9jZQ7zJKaWpobgcCocF02WC1QrjbKUiGf72SV_cdTx4nGN0xI0qJWpCCLlbitVtm-
sppkWYySFJUnmc_Mtpua2lcfl_g-k0k0sQrhA/file# [following]
--2024-02-19 13:00:04--  https://uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercont
ent.com/cd/0/inline/CNmmmw9wTypFuF6p7dk3UgcIXG7N-
9jZQ7zJKaWpobgcCocF02WC1QrjbKUiGf72SV_cdTx4nGN0xI0qJWpCCLlbitVtm-
sppkWYySFJUnmc_Mtpua2lcfl_g-k0k0sQrhA/file
Resolving uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com
(uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com)…
2620:100:6020:15::a27d:400f, 162.125.64.15
Connecting to uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com (uce9d8a7f9
26b4e7d1a615152f19.dl.dropboxusercontent.com)|2620:100:6020:15::a27d:400f|:443..
. failed: Operation timed out.
Connecting to uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com
(uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercontent.com)|162.125.64.15|:443…
connected.
HTTP request sent, awaiting response… 302 Found
Location: /cd/0/inline2/CNkV-VGONQHTiH1-
2si29_szQe4X20h0_szlEVfJFnhSkoUvDLDuP4zCQKIvOYCQxxGo-z8FK7-QH25JU7dD7JR-EVUDoVuI
xR_tmuwFawq0shZwcB0mjEZX_ooeqLtNtGCXPUxAGYU0A0oVL8AWXgRHEozWATSzNAgZlAthKhnULAbT
rM95CUKqeSu5_ePKL9cmIB-nXHW6hgW4CG87MbNf9ooJsN3gfHz3Wj-
z7EpDLLydQ77J5H6L4pav7J7RKgz3aKSp8FBITaXSu8Q3d_h89nuS0yChIY2UUma2Dl-
Y9xP1ppyUKoL1C-jmmG2p1znRv25x0o0r6lCZxMliQjsosn3wmiopZTwpEITXT-92PQ/file
[following]
--2024-02-19 13:01:20--  https://uce9d8a7f926b4e7d1a615152f19.dl.dropboxusercont
ent.com/cd/0/inline2/CNkV-VGONQHTiH1-
2si29_szQe4X20h0_szlEVfJFnhSkoUvDLDuP4zCQKIvOYCQxxGo-z8FK7-QH25JU7dD7JR-EVUDoVuI
xR_tmuwFawq0shZwcB0mjEZX_ooeqLtNtGCXPUxAGYU0A0oVL8AWXgRHEozWATSzNAgZlAthKhnULAbT
rM95CUKqeSu5_ePKL9cmIB-nXHW6hgW4CG87MbNf9ooJsN3gfHz3Wj-
z7EpDLLydQ77J5H6L4pav7J7RKgz3aKSp8FBITaXSu8Q3d_h89nuS0yChIY2UUma2Dl-
Y9xP1ppyUKoL1C-jmmG2p1znRv25x0o0r6lCZxMliQjsosn3wmiopZTwpEITXT-92PQ/file
Reusing existing connection to
uce9d8a7f926b4e7d1a615152f19.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  13.4MB/s    in 0.7s
```

```
2024-02-19 13:01:21 (13.4 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:

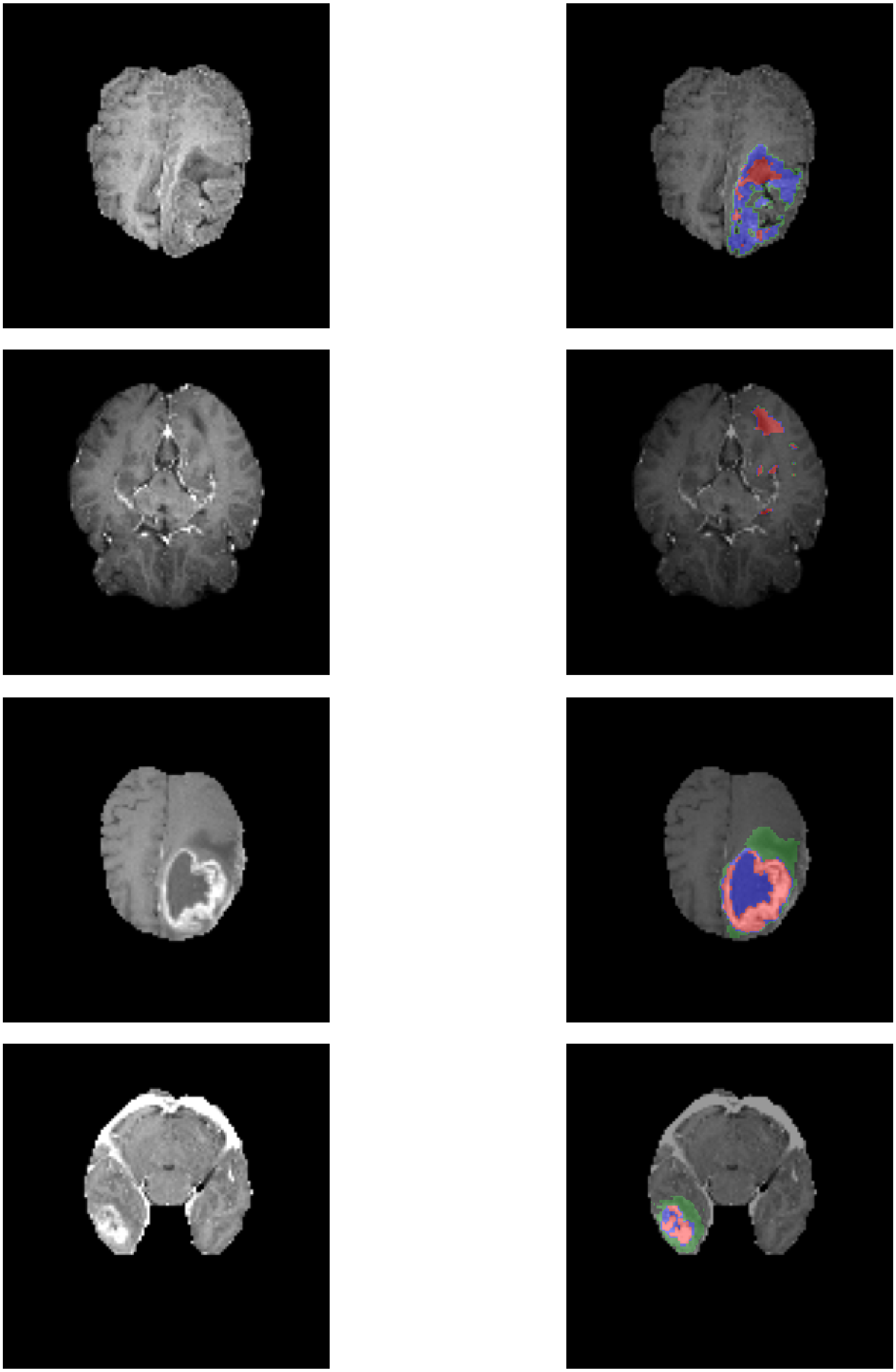
`cmap = colors.ListedColormap(['black', 'green', 'blue', 'red'])`

[107]:

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

def select_random_pairs(training_dir, label_dir, num_pairs=4):
    files = os.listdir(training_dir)
    selected_files = random.sample(files, num_pairs)

    images = []
    labels = []
    for f in selected_files:
      images.append(os.path.join(training_dir, f))
      labels.append(os.path.join(label_dir, f))

    return images, labels

def visualize_with_labels(images, labels):
    _, axs = plt.subplots(4, 2, figsize=(10, 10))

    for i, (image_path, label_path) in enumerate(zip(images, labels)):
        image = imageio.imread(image_path)
        label = imageio.imread(label_path)

        # display the original image in the first column
        axs[i, 0].imshow(image, cmap='gray')
        axs[i, 0].axis('off')

        # display the overlayed image in the second column
        axs[i, 1].imshow(image, cmap='gray')  # first display the image in␣
↪grayscale
        axs[i, 1].imshow(label, cmap=cmap, alpha=0.4)  # then overlay the label
        axs[i, 1].axis('off')

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

```python
training_dir = 'Task01_BrainTumour_2D/training_images'
label_dir = 'Task01_BrainTumour_2D/training_labels'

images, labels = select_random_pairs(training_dir, label_dir, 4)

# Visualize
visualize_with_labels(images, labels)
```

```
/var/folders/g8/tv0n_c7d0d77hmpf9kw7mvrw0000gn/T/ipykernel_65185/1939092974.py:1
9: 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(image_path)
/var/folders/g8/tv0n_c7d0d77hmpf9kw7mvrw0000gn/T/ipykernel_65185/1939092974.py:2
0: 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_path)
```



## 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.

[108]:
```
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
        # print(self.images[0].shape, self.labels[0].shape)
        images, labels = [np.array(t)for t in list(zip(*random.
↪sample(list(self), batch_size)))]
        image_arr = np.stack(images)[:, np.newaxis, :, :]
        label_arr = np.stack(labels)
        return image_arr, label_arr
```

## 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.

[109]:
```
""" 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(
```

```python
            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
        self.upconv3 = nn.Sequential(
            nn.ConvTranspose2d(n, int(n / 2), kernel_size=2, stride=2),
            nn.ReLU()
        )
        self.dec_conv3 = 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()
        )

        n //= 2  # 64
        self.upconv2 = nn.Sequential(
            nn.ConvTranspose2d(int(n), int(n / 2), kernel_size=2, stride=2),
            nn.ReLU()
        )
```

```
        self.dec_conv2 = 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()
        )

        n //= 2  # 32
        self.upconv1 = nn.Sequential(
            nn.ConvTranspose2d(int(n), int(n / 2), kernel_size=2, stride=2),
            nn.ReLU()
        )
        self.dec_conv1 = 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()
        )

        n //= 2  # 16
        self.final = nn.Conv2d(int(n), output_channel, kernel_size=1)

    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
        x = self.upconv3(x)
        x = torch.cat((x, conv3_skip), dim=1)
        x = self.dec_conv3(x)
```

```
        x = self.upconv2(x)
        x = torch.cat((x, conv2_skip), dim=1)
        x = self.dec_conv2(x)

        x = self.upconv1(x)
        x = torch.cat((x, conv1_skip), dim=1)
        x = self.dec_conv1(x)

        x = self.final(x)

        return x
```

### 1.8 4. Train the segmentation model.

[110]:
```
# 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('Iter {0}, Training 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('Iter {0}, Test Loss: {1:.4f}'.format(it, test_loss.item()))
            ### End of your code ###

    # Save the model
    if it % 5000 == 0:
        print("Saving the model")
```

```
        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))
```

```
/var/folders/g8/tv0n_c7d0d77hmpf9kw7mvrw0000gn/T/ipykernel_65185/1515733613.py:2
3: 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))
/var/folders/g8/tv0n_c7d0d77hmpf9kw7mvrw0000gn/T/ipykernel_65185/1515733613.py:2
9: 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)
```

```
Device: cpu
(output below manually shortened)
Iter 1, Training Loss: 1.4800
Iter 100, Training Loss: 0.4105
Iter 100, Test Loss: 0.4584
Iter 200, Training Loss: 0.1598
Iter 200, Test Loss: 0.1798
Iter 300, Training Loss: 0.0912
Iter 300, Test Loss: 0.2603
Iter 400, Training Loss: 0.0735
Iter 400, Test Loss: 0.0948
Iter 500, Training Loss: 0.0795
Iter 500, Test Loss: 0.0655
Iter 600, Training Loss: 0.0561
Iter 600, Test Loss: 0.0616
Iter 700, Training Loss: 0.0495
Iter 700, Test Loss: 0.0416
Iter 800, Training Loss: 0.0350
Iter 800, Test Loss: 0.0386
Iter 900, Training Loss: 0.0519
Iter 900, Test Loss: 0.0566
Iter 1000, Training Loss: 0.0496
Iter 1000, Test Loss: 0.0560
Iter 1100, Training Loss: 0.0484
Iter 1100, Test Loss: 0.0532
Iter 1200, Training Loss: 0.0368
Iter 1200, Test Loss: 0.0406
Iter 1300, Training Loss: 0.0446
Iter 1300, Test Loss: 0.0452
Iter 1400, Training Loss: 0.0450
Iter 1400, Test Loss: 0.0611
```

```
Iter 1500, Training Loss: 0.0341
Iter 1500, Test Loss: 0.0375
Iter 1600, Training Loss: 0.0353
Iter 1600, Test Loss: 0.0414
Iter 1700, Training Loss: 0.0203
Iter 1700, Test Loss: 0.0434
Iter 1800, Training Loss: 0.0309
Iter 1800, Test Loss: 0.0439
Iter 1900, Training Loss: 0.0344
Iter 1900, Test Loss: 0.0433
Iter 2000, Training Loss: 0.0269
Iter 2000, Test Loss: 0.0371
Iter 2100, Training Loss: 0.0243
Iter 2100, Test Loss: 0.0430
Iter 2200, Training Loss: 0.0317
Iter 2200, Test Loss: 0.0675
Iter 2300, Training Loss: 0.0281
Iter 2300, Test Loss: 0.0166
Iter 2400, Training Loss: 0.0189
Iter 2400, Test Loss: 0.0391
Iter 2500, Training Loss: 0.0195
Iter 2500, Test Loss: 0.0316
Iter 2600, Training Loss: 0.0237
Iter 2600, Test Loss: 0.0317
Iter 2700, Training Loss: 0.0195
Iter 2700, Test Loss: 0.0450
Iter 2800, Training Loss: 0.0179
Iter 2800, Test Loss: 0.0272
Iter 2900, Training Loss: 0.0277
Iter 2900, Test Loss: 0.0402
Iter 3000, Training Loss: 0.0138
Iter 3000, Test Loss: 0.0261
Iter 3100, Training Loss: 0.0158
Iter 3100, Test Loss: 0.0419
Iter 3200, Training Loss: 0.0105
Iter 3200, Test Loss: 0.0633
Iter 3300, Training Loss: 0.0133
Iter 3300, Test Loss: 0.0373
Iter 3400, Training Loss: 0.0125
Iter 3400, Test Loss: 0.0270
Iter 3500, Training Loss: 0.0141
Iter 3500, Test Loss: 0.0571
Iter 3600, Training Loss: 0.0202
Iter 3600, Test Loss: 0.0314
Iter 3700, Training Loss: 0.0122
Iter 3700, Test Loss: 0.0432
Iter 3800, Training Loss: 0.0227
Iter 3800, Test Loss: 0.0521
```

```
Iter 3900, Training Loss: 0.0174
Iter 3900, Test Loss: 0.0565
Iter 4000, Training Loss: 0.0171
Iter 4000, Test Loss: 0.0427
Iter 4100, Training Loss: 0.0144
Iter 4100, Test Loss: 0.0387
Iter 4200, Training Loss: 0.0142
Iter 4200, Test Loss: 0.0286
Iter 4300, Training Loss: 0.0165
Iter 4300, Test Loss: 0.0196
Iter 4400, Training Loss: 0.0127
Iter 4400, Test Loss: 0.0332
Iter 4500, Training Loss: 0.0146
Iter 4500, Test Loss: 0.0293
Iter 4600, Training Loss: 0.0187
Iter 4600, Test Loss: 0.0363
Iter 4700, Training Loss: 0.0141
Iter 4700, Test Loss: 0.0255
Iter 4800, Training Loss: 0.0079
Iter 4800, Test Loss: 0.0624
Iter 4900, Training Loss: 0.0141
Iter 4900, Test Loss: 0.0240
Iter 5000, Training Loss: 0.0152
Iter 5000, Test Loss: 0.0413
Iter 5100, Training Loss: 0.0086
Iter 5100, Test Loss: 0.0482
Iter 5200, Training Loss: 0.0132
Iter 5200, Test Loss: 0.0314
Iter 5300, Training Loss: 0.0123
Iter 5300, Test Loss: 0.0130
Iter 5400, Training Loss: 0.0086
Iter 5400, Test Loss: 0.0390
Iter 5500, Training Loss: 0.0163
Iter 5500, Test Loss: 0.0482
Iter 5600, Training Loss: 0.0134
Iter 5600, Test Loss: 0.0422
Iter 5700, Training Loss: 0.0117
Iter 5700, Test Loss: 0.0441
Iter 5800, Training Loss: 0.0132
Iter 5800, Test Loss: 0.0530
Iter 5900, Training Loss: 0.0098
Iter 5900, Test Loss: 0.0188
Iter 6000, Training Loss: 0.0084
Iter 6000, Test Loss: 0.0528
Iter 6100, Training Loss: 0.0099
Iter 6100, Test Loss: 0.0311
Iter 6200, Training Loss: 0.0142
Iter 6200, Test Loss: 0.0411
```

```
Iter 6300, Training Loss: 0.0091
Iter 6300, Test Loss: 0.0265
Iter 6400, Training Loss: 0.0128
Iter 6400, Test Loss: 0.0505
Iter 6500, Training Loss: 0.0119
Iter 6500, Test Loss: 0.0391
Iter 6600, Training Loss: 0.0103
Iter 6600, Test Loss: 0.0313
Iter 6700, Training Loss: 0.0114
Iter 6700, Test Loss: 0.0501
Iter 6800, Training Loss: 0.0092
Iter 6800, Test Loss: 0.0455
Iter 6900, Training Loss: 0.0103
Iter 6900, Test Loss: 0.0468
Iter 7000, Training Loss: 0.0100
Iter 7000, Test Loss: 0.0473
Iter 7100, Training Loss: 0.0067
Iter 7100, Test Loss: 0.0661
Iter 7200, Training Loss: 0.0091
Iter 7200, Test Loss: 0.0451
Iter 7300, Training Loss: 0.0100
Iter 7300, Test Loss: 0.0580
Iter 7400, Training Loss: 0.0078
Iter 7400, Test Loss: 0.0533
Iter 7500, Training Loss: 0.0107
Iter 7500, Test Loss: 0.0440
Iter 7600, Training Loss: 0.0060
Iter 7600, Test Loss: 0.0550
Iter 7700, Training Loss: 0.0094
Iter 7700, Test Loss: 0.0667
Iter 7800, Training Loss: 0.0103
Iter 7800, Test Loss: 0.0994
Iter 7900, Training Loss: 0.0132
Iter 7900, Test Loss: 0.0244
Iter 8000, Training Loss: 0.0090
Iter 8000, Test Loss: 0.0637
Iter 8100, Training Loss: 0.0118
Iter 8100, Test Loss: 0.0596
Iter 8200, Training Loss: 0.0094
Iter 8200, Test Loss: 0.0455
Iter 8300, Training Loss: 0.0077
Iter 8300, Test Loss: 0.0424
Iter 8400, Training Loss: 0.0064
Iter 8400, Test Loss: 0.0435
Iter 8500, Training Loss: 0.0151
Iter 8500, Test Loss: 0.0461
Iter 8600, Training Loss: 0.0069
Iter 8600, Test Loss: 0.0388
```

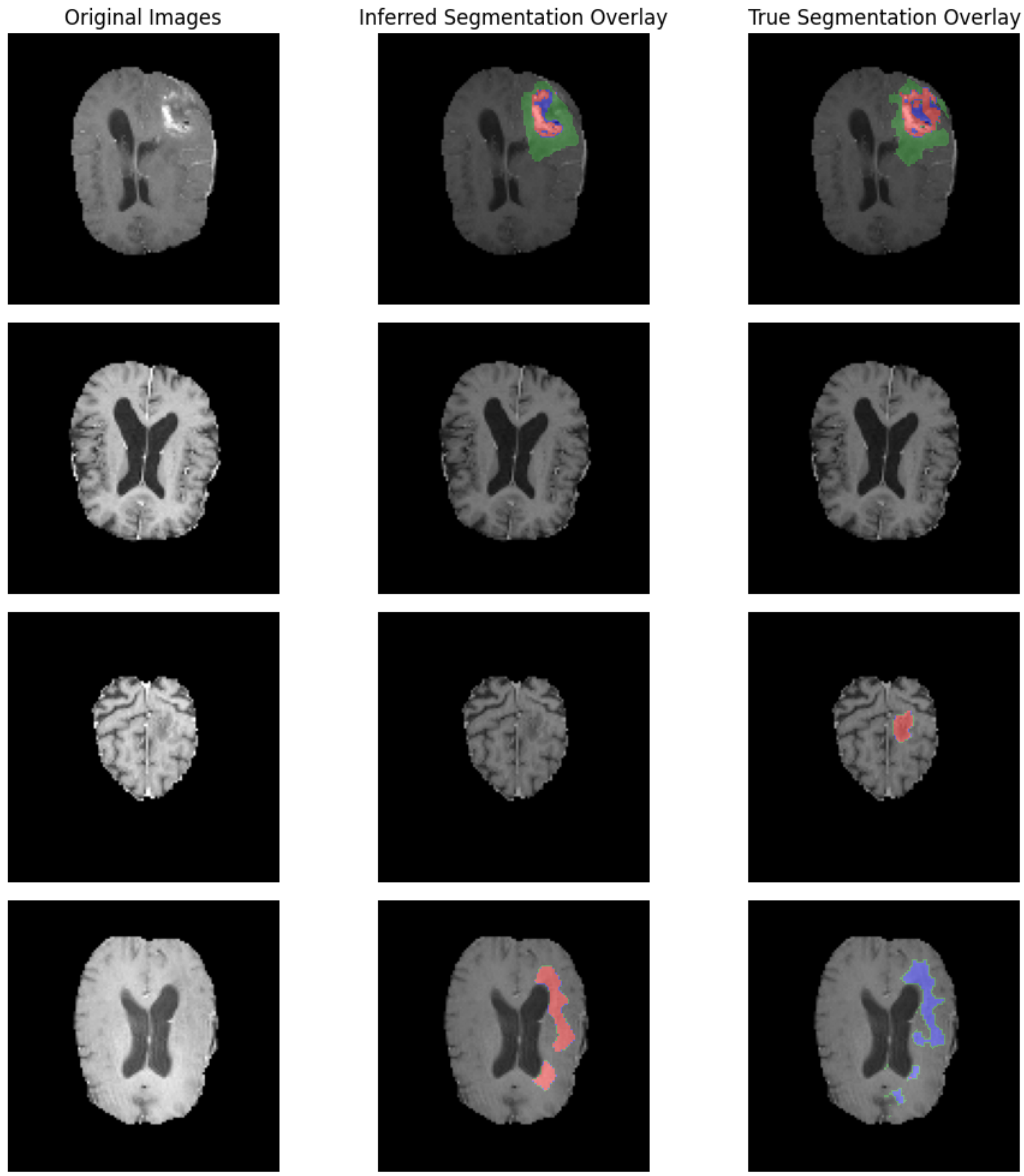
```
Iter 8700, Training Loss: 0.0101
Iter 8700, Test Loss: 0.0373
Iter 8800, Training Loss: 0.0100
Iter 8800, Test Loss: 0.0573
Iter 8900, Training Loss: 0.0081
Iter 8900, Test Loss: 0.0484
Iter 9000, Training Loss: 0.0084
Iter 9000, Test Loss: 0.0312
Iter 9100, Training Loss: 0.0103
Iter 9100, Test Loss: 0.0539
Iter 9200, Training Loss: 0.0121
Iter 9200, Test Loss: 0.0365
Iter 9300, Training Loss: 0.0096
Iter 9300, Test Loss: 0.0593
Iter 9400, Training Loss: 0.0125
Iter 9400, Test Loss: 0.0540
Iter 9500, Training Loss: 0.0117
Iter 9500, Test Loss: 0.0519
Iter 9600, Training Loss: 0.0096
Iter 9600, Test Loss: 0.0570
Iter 9700, Training Loss: 0.0058
Iter 9700, Test Loss: 0.0970
Iter 9800, Training Loss: 0.0092
Iter 9800, Test Loss: 0.0309
Iter 9900, Training Loss: 0.0039
Iter 9900, Test Loss: 0.0681
Iter 10000, Training Loss: 0.0097
Iter 10000, Test Loss: 0.0525
Saving the model
Training took 15344.810s 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.

[120]:
```
def visualize_with_labels(display_data_list):
    _, axs = plt.subplots(4, 3, figsize=(10, 10))

    axs[0, 0].set_title('Original Images')
    axs[0, 1].set_title('Inferred Segmentation Overlay')
    axs[0, 2].set_title('True Segmentation Overlay')

    for i, (image, inferred_labels, true_labels) in enumerate(display_data_list):
        axs[i, 0].imshow(image, cmap='gray')
        axs[i, 0].axis('off')
```

```
        axs[i, 1].imshow(image, cmap='gray')
        axs[i, 1].imshow(inferred_labels, cmap=cmap, alpha=0.4)
        axs[i, 1].axis('off')

        axs[i, 2].imshow(image, cmap='gray')
        axs[i, 2].imshow(true_labels, cmap=cmap, alpha=0.4)
        axs[i, 2].axis('off')

    plt.tight_layout()
    plt.show()

model = UNet(input_channel=1, output_channel=num_class, num_filter=16)
model.load_state_dict(torch.load('saved_models/model_10000.pt'))
model.eval()

def evaluate_image(image):
    with torch.no_grad():
        torch_image = torch.from_numpy(image)
        device_image = torch_image.to(device, dtype=torch.float32)
        logits = model(device_image)
        probabilities = F.softmax(logits, dim=1)
        label_map = torch.argmax(probabilities, dim=1)
        return label_map

def test_data_to_displayable(image, inferred_map, map):
    return image.squeeze(), inferred_map.squeeze().cpu().numpy(), map.squeeze()

display_data_list = []
for i in range(4):
    test_image, test_label = test_set.get_random_batch(1)
    inferred_map = evaluate_image(test_image)
    display_data_list.append(test_data_to_displayable(test_image, inferred_map,
 ↪test_label))

visualize_with_labels(display_data_list)
```




## 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 trained model appears to usually do segmentation fine, but it appears to have a fairly high false positive rate. To make sure it performs well in real-world scenarios, we should test it on different datasets and use cross-validation to confirm its reliability, and likely use a larger dataset. I think it's also important to add explanation features so people can understand how it makes decisions, and to ensure it follows medical software regulations, and set up ongoing performance monitoring. Running a few human studies and doing statistical tests to check how effective it is would also be good. These steps will help make the model more reliable, build trust with healthcare professionals, and meet the requirements of clinical environments.

```
[ ]:
```