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 [2]:
```
# 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

I commented out wget because i received the following error message upon running it:

```
'wget' is not recognized as an internal or external command, operable program or batch file.
```

I instead just manually downloaded the files.

In [3]:
```
# 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()
```

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

Suggested colour map for brain MR image:

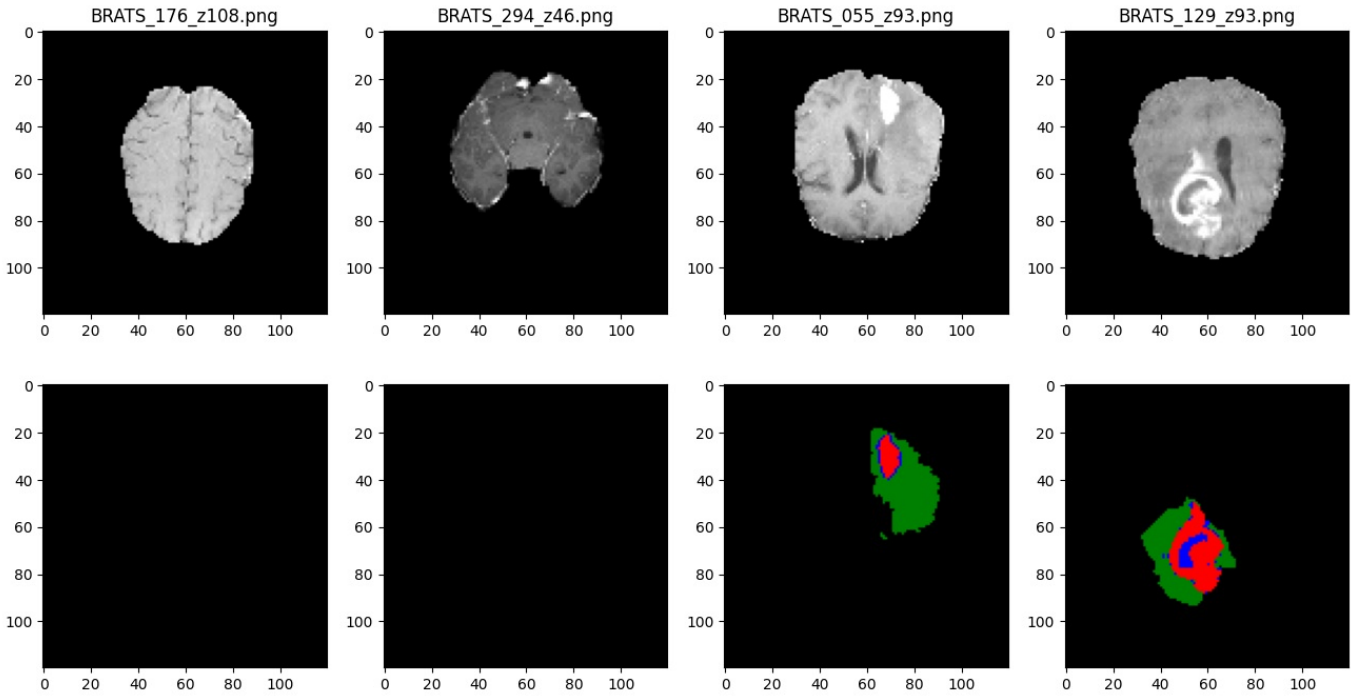
```
cmap = 'gray'
```

Suggested colour map for segmentation map:

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

In [4]:
```
### Insert your code ###
rng = np.random.default_rng()
image_names = rng.permutation(os.listdir("Task01_BrainTumour_2D/training_images"))
for i in range(4):
    image = imageio.imread(os.path.join("Task01_BrainTumour_2D/training_images", image_names[i]))
    labels = imageio.imread(os.path.join("Task01_BrainTumour_2D/training_labels", image_names[i]))
    plt.subplot(2,4,i+1)
    plt.title(f"{image_names[i]}")
    plt.imshow(image, cmap='gray')
    plt.subplot(2,4,i+5)
    plt.imshow(labels, cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
plt.gcf().set_size_inches(16, 8)
### End of your code ###
```

```
C:\Users\jacob\AppData\Local\Temp\ipykernel_2108\703456415.py:5: DeprecationWarning: Starting with ImageIO v3 th
e behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and make this wa
rning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  image = imageio.imread(os.path.join("Task01_BrainTumour_2D/training_images", image_names[i]))
C:\Users\jacob\AppData\Local\Temp\ipykernel_2108\703456415.py:6: DeprecationWarning: Starting with ImageIO v3 th
e behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and make this wa
rning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  labels = imageio.imread(os.path.join("Task01_BrainTumour_2D/training_labels", image_names[i]))
```



2. Implement a dataset class.

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

In [25]:
```
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, rng=np.random.default_rng()):
        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.append(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.append(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 = rng.permutation(len(self.images))[:batch_size]
        images = np.expand_dims(np.array([normalise_intensity(image) for image in np.array(self.images)[indices
        labels = np.array(self.labels)[indices]

        ### 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 [147…
```
""" 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 *= 2  # 256

        self.conv5 = nn.Sequential( # Bottom layer of U is technically Encoder
            nn.Conv2d(n//2, n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.ConvTranspose2d(n, n//2, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n//2),
            nn.ReLU(),
        )

        n //= 2 # 128

        self.conv6 = nn.Sequential(
            nn.Conv2d(n*2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.ConvTranspose2d(n, n//2, kernel_size=3, stride=2, output_padding=1, padding=1),
            nn.BatchNorm2d(n//2),
            nn.ReLU(),
        )

        n //= 2 # 64

        self.conv7 = nn.Sequential(
            nn.Conv2d(n*2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.ConvTranspose2d(n, n//2, kernel_size=3, stride=2, output_padding=1, padding=1),
            nn.BatchNorm2d(n//2),
            nn.ReLU(),
        )

        n //= 2 # 32

        self.conv8 = nn.Sequential(
            nn.Conv2d(n*2, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.ConvTranspose2d(n, n//2, kernel_size=3, stride=2, output_padding=1, padding=1),
            nn.BatchNorm2d(n//2),
            nn.ReLU(),
```

```
        )

        n //= 2 # 16

        self.conv9 = 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(),
            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 ###

        conv4_skip = x

        x = self.conv5(x)

        x = self.conv6(torch.cat((conv4_skip, x), dim=1))

        x = self.conv7(torch.cat((conv3_skip, x), dim=1))

        x = self.conv8(torch.cat((conv2_skip, x), dim=1))

        x = self.conv9(torch.cat((conv1_skip, x), dim=1))

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

4. Train the segmentation model.

In [173…

```
# 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-4)

# Segmentation loss
criterion = nn.CrossEntropyLoss(torch.tensor([1,5,10,15]).to(device,dtype=torch.float32))

# 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()
```

```python
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 ###

    loss = criterion(logits, labels)
    loss.backward()
    optimizer.step()

    if it % 50 == 0:
        print(f"[{it}] - Loss -> {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 ###

            # Get a batch of images and labels
            eval_images, eval_labels = test_set.get_random_batch(eval_batch_size)
            eval_images, eval_labels = torch.from_numpy(eval_images), torch.from_numpy(eval_labels)
            eval_images, eval_labels = eval_images.to(device, dtype=torch.float32), eval_labels.to(device, dtyp
            eval_logits = model(eval_images)
            eval_loss = criterion(eval_logits, eval_labels)
            print(f"[{it}] - Eval Loss -> {eval_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
C:\Users\jacob\AppData\Local\Temp\ipykernel_2108\2948606474.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))
C:\Users\jacob\AppData\Local\Temp\ipykernel_2108\2948606474.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)
[50] - Loss -> 0.9078689813613892
[100] - Loss -> 0.600203275680542
[100] - Eval Loss -> 0.5460102558135986
[150] - Loss -> 0.3934304714202881
[200] - Loss -> 0.3369516432285309
[200] - Eval Loss -> 0.4201222062110901
[250] - Loss -> 0.22408601641654968
[300] - Loss -> 0.30022749304771423
[300] - Eval Loss -> 0.47650858759880066
[350] - Loss -> 0.3103870749473572
[400] - Loss -> 0.3903177082538605
[400] - Eval Loss -> 0.4295807480812073
[450] - Loss -> 0.3733900785446167
[500] - Loss -> 0.3093791604042053
[500] - Eval Loss -> 0.4449636936187744
[550] - Loss -> 0.3385361135005951
[600] - Loss -> 0.24997729063034058
[600] - Eval Loss -> 0.32405978441238403
[650] - Loss -> 0.33903810381889343
[700] - Loss -> 0.234308660030365
[700] - Eval Loss -> 0.3544187843799591
[750] - Loss -> 0.40296027064323425
[800] - Loss -> 0.3372868001461029
[800] - Eval Loss -> 0.2995529770851135
[850] - Loss -> 0.39617305994033813
[900] - Loss -> 0.27369067072868347
[900] - Eval Loss -> 0.3123425543308258
[950] - Loss -> 0.23511353135108948
```

```
[1000] - Loss -> 0.35015246272087097
[1000] - Eval Loss -> 0.3011004626750946
[1050] - Loss -> 0.3702523112297058
[1100] - Loss -> 0.3226619362831116
[1100] - Eval Loss -> 0.3939959704875946
[1150] - Loss -> 0.37964656949043274
[1200] - Loss -> 0.35853812098503113
[1200] - Eval Loss -> 0.4292868375778198
[1250] - Loss -> 0.43213364481925964
[1300] - Loss -> 0.21538683772087097
[1300] - Eval Loss -> 0.2301318198442459
[1350] - Loss -> 0.3156733512878418
[1400] - Loss -> 0.3964978754520416
[1400] - Eval Loss -> 0.29371583461761475
[1450] - Loss -> 0.3510493040084839
[1500] - Loss -> 0.23009687662124634
[1500] - Eval Loss -> 0.27420178055763245
[1550] - Loss -> 0.25033843517303467
[1600] - Loss -> 0.3420892357826233
[1600] - Eval Loss -> 0.32541194558143616
[1650] - Loss -> 0.26733434200286865
[1700] - Loss -> 0.23886679112911224
[1700] - Eval Loss -> 0.2640361487865448
[1750] - Loss -> 0.19458995759487152
[1800] - Loss -> 0.35796913504600525
[1800] - Eval Loss -> 0.32382968068122864
[1850] - Loss -> 0.21976418793201447
[1900] - Loss -> 0.27632761001586914
[1900] - Eval Loss -> 0.30700576305389404
[1950] - Loss -> 0.19837632775306702
[2000] - Loss -> 0.32823026180267334
[2000] - Eval Loss -> 0.2592877149581909
[2050] - Loss -> 0.33778390288352966
[2100] - Loss -> 0.2674245536327362
[2100] - Eval Loss -> 0.3102301061153412
[2150] - Loss -> 0.24632000923156738
[2200] - Loss -> 0.22802089154720306
[2200] - Eval Loss -> 0.3166395425796509
[2250] - Loss -> 0.3184378445148468
[2300] - Loss -> 0.23133473098278046
[2300] - Eval Loss -> 0.4268210530281067
[2350] - Loss -> 0.2453155517578125
[2400] - Loss -> 0.40819409489631653
[2400] - Eval Loss -> 0.370145708322525
[2450] - Loss -> 0.22412194311618805
[2500] - Loss -> 0.3298211693763733
[2500] - Eval Loss -> 0.26931917667388916
[2550] - Loss -> 0.2566232979297638
[2600] - Loss -> 0.2960500717163086
[2600] - Eval Loss -> 0.279940128326416
[2650] - Loss -> 0.2738407254219055
[2700] - Loss -> 0.3901500403881073
[2700] - Eval Loss -> 0.3014586567878723
[2750] - Loss -> 0.2444203794002533
[2800] - Loss -> 0.328255832195282
[2800] - Eval Loss -> 0.38756483793258667
[2850] - Loss -> 0.3098613917827606
[2900] - Loss -> 0.2188030332326889
[2900] - Eval Loss -> 0.29345229268074036
[2950] - Loss -> 0.2539261281490326
[3000] - Loss -> 0.3369438052177429
[3000] - Eval Loss -> 0.2467784881591797
[3050] - Loss -> 0.2574755549430847
[3100] - Loss -> 0.37501636147499084
[3100] - Eval Loss -> 0.24449612200260162
[3150] - Loss -> 0.35654690861701965
[3200] - Loss -> 0.22100767493247986
[3200] - Eval Loss -> 0.2432183027267456
[3250] - Loss -> 0.3024243712425232
[3300] - Loss -> 0.2966042757034302
[3300] - Eval Loss -> 0.36454230546951294
[3350] - Loss -> 0.2589380145072937
[3400] - Loss -> 0.22748254239559174
[3400] - Eval Loss -> 0.34221091866493225
[3450] - Loss -> 0.39838749170303345
[3500] - Loss -> 0.2483009696006775
[3500] - Eval Loss -> 0.3274378776550293
[3550] - Loss -> 0.2850312888622284
[3600] - Loss -> 0.3495001494884491
[3600] - Eval Loss -> 0.2908088266849518
[3650] - Loss -> 0.24290825426578522
[3700] - Loss -> 0.35600361227989197
[3700] - Eval Loss -> 0.3430425822734833
```

```
[3750] - Loss -> 0.3294234275817871
[3800] - Loss -> 0.34513768553733826
[3800] - Eval Loss -> 0.3104008734226227
[3850] - Loss -> 0.31801682710647583
[3900] - Loss -> 0.1873161941766739
[3900] - Eval Loss -> 0.27813005447387695
[3950] - Loss -> 0.17356102168560028
[4000] - Loss -> 0.2119390368461609
[4000] - Eval Loss -> 0.32992270588874817
[4050] - Loss -> 0.26271340250968933
[4100] - Loss -> 0.20723646879196167
[4100] - Eval Loss -> 0.30580341815948486
[4150] - Loss -> 0.30881267786026
[4200] - Loss -> 0.2714427709579468
[4200] - Eval Loss -> 0.27950936555862427
[4250] - Loss -> 0.3256967067718506
[4300] - Loss -> 0.24193024635314941
[4300] - Eval Loss -> 0.36189281940460205
[4350] - Loss -> 0.2120678871870041
[4400] - Loss -> 0.30762919783592224
[4400] - Eval Loss -> 0.31437698006629944
[4450] - Loss -> 0.2610034644603729
[4500] - Loss -> 0.2707574665546417
[4500] - Eval Loss -> 0.2793252468109131
[4550] - Loss -> 0.2231842428445816
[4600] - Loss -> 0.2656483054161072
[4600] - Eval Loss -> 0.28084999322891235
[4650] - Loss -> 0.30895286798477173
[4700] - Loss -> 0.2302161604166031
[4700] - Eval Loss -> 0.4107588231563568
[4750] - Loss -> 0.28727346658706665
[4800] - Loss -> 0.27710869908332825
[4800] - Eval Loss -> 0.3641814887523651
[4850] - Loss -> 0.22967761754989624
[4900] - Loss -> 0.2537505030632019
[4900] - Eval Loss -> 0.4740613102912903
[4950] - Loss -> 0.25926604866981506
[5000] - Loss -> 0.21619822084903717
[5000] - Eval Loss -> 0.24281810224056244
[5050] - Loss -> 0.2957700490951538
[5100] - Loss -> 0.21026429533958435
[5100] - Eval Loss -> 0.24006260931491852
[5150] - Loss -> 0.14612849056720734
[5200] - Loss -> 0.23512664437294006
[5200] - Eval Loss -> 0.34919053316116333
[5250] - Loss -> 0.31608715653419495
[5300] - Loss -> 0.27130916714668274
[5300] - Eval Loss -> 0.18993766605854034
[5350] - Loss -> 0.2475837767124176
[5400] - Loss -> 0.24420984089374542
[5400] - Eval Loss -> 0.28700730204582214
[5450] - Loss -> 0.21432416141033173
[5500] - Loss -> 0.242283895611763
[5500] - Eval Loss -> 0.25950586795806885
[5550] - Loss -> 0.3306729793548584
[5600] - Loss -> 0.2374756783246994
[5600] - Eval Loss -> 0.22811216115951538
[5650] - Loss -> 0.3629899322986603
[5700] - Loss -> 0.3476293385028839
[5700] - Eval Loss -> 0.19378766417503357
[5750] - Loss -> 0.25150948762893677
[5800] - Loss -> 0.27015256881713867
[5800] - Eval Loss -> 0.2867007553577423
[5850] - Loss -> 0.26554009318351746
[5900] - Loss -> 0.37110915780067444
[5900] - Eval Loss -> 0.24083723127841955
[5950] - Loss -> 0.2761448323726654
[6000] - Loss -> 0.2360849678516388
[6000] - Eval Loss -> 0.22424496710300446
[6050] - Loss -> 0.24678397178649902
[6100] - Loss -> 0.3320077657699585
[6100] - Eval Loss -> 0.21063622832298828
[6150] - Loss -> 0.2810561954975128
[6200] - Loss -> 0.22572341561317444
[6200] - Eval Loss -> 0.197698175907135
[6250] - Loss -> 0.21998187899589539
[6300] - Loss -> 0.24043744802474976
[6300] - Eval Loss -> 0.2589252293109894
[6350] - Loss -> 0.28626462817192078
[6400] - Loss -> 0.262997567653656
[6400] - Eval Loss -> 0.26829835772514343
[6450] - Loss -> 0.28661343455314636
[6500] - Loss -> 0.18426558375358582
```

```
[6500] - Eval Loss -> 0.2461509257555008
[6550] - Loss -> 0.30783963203430176
[6600] - Loss -> 0.23231399059295654
[6600] - Eval Loss -> 0.21367943286895752
[6650] - Loss -> 0.32983818650245667
[6700] - Loss -> 0.22917205095291138
[6700] - Eval Loss -> 0.19715000689029694
[6750] - Loss -> 0.25985267758369446
[6800] - Loss -> 0.19487395882606506
[6800] - Eval Loss -> 0.3342934250831604
[6850] - Loss -> 0.2680814862251282
[6900] - Loss -> 0.2943754196166992
[6900] - Eval Loss -> 0.3035131096839905
[6950] - Loss -> 0.29934829473495483
[7000] - Loss -> 0.2811804413795471
[7000] - Eval Loss -> 0.27826395630836487
[7050] - Loss -> 0.20498566329479218
[7100] - Loss -> 0.3133428990840912
[7100] - Eval Loss -> 0.23626041412353516
[7150] - Loss -> 0.3074783980846405
[7200] - Loss -> 0.27944859862327576
[7200] - Eval Loss -> 0.2732231914997101
[7250] - Loss -> 0.19963745772838593
[7300] - Loss -> 0.34755101799964905
[7300] - Eval Loss -> 0.3297474682331085
[7350] - Loss -> 0.1491934210062027
[7400] - Loss -> 0.22579897940158844
[7400] - Eval Loss -> 0.26823535561561584
[7450] - Loss -> 0.3007563054561615
[7500] - Loss -> 0.3646399974822998
[7500] - Eval Loss -> 0.17957134544849396
[7550] - Loss -> 0.15690606832504272
[7600] - Loss -> 0.34534725546836853
[7600] - Eval Loss -> 0.22455380856990814
[7650] - Loss -> 0.257409930229187
[7700] - Loss -> 0.27979645133018494
[7700] - Eval Loss -> 0.25085747241973877
[7750] - Loss -> 0.30243393778800964
[7800] - Loss -> 0.21572594344615936
[7800] - Eval Loss -> 0.2867509722709656
[7850] - Loss -> 0.13600413501262665
[7900] - Loss -> 0.3229651153087616
[7900] - Eval Loss -> 0.1781424731016159
[7950] - Loss -> 0.22719517350196838
[8000] - Loss -> 0.22595292329788208
[8000] - Eval Loss -> 0.21718494594097137
[8050] - Loss -> 0.2560312747955322
[8100] - Loss -> 0.17305614054203033
[8100] - Eval Loss -> 0.29818612337112427
[8150] - Loss -> 0.14251607656478882
[8200] - Loss -> 0.3044263422489166
[8200] - Eval Loss -> 0.22203579545021057
[8250] - Loss -> 0.32683247327804565
[8300] - Loss -> 0.2469179630279541
[8300] - Eval Loss -> 0.3344366252422333
[8350] - Loss -> 0.29501038789749146
[8400] - Loss -> 0.25183355808258057
[8400] - Eval Loss -> 0.2825014293193817
[8450] - Loss -> 0.31546780467033386
[8500] - Loss -> 0.18936215341091156
[8500] - Eval Loss -> 0.3293752074241638
[8550] - Loss -> 0.337074875831604
[8600] - Loss -> 0.25514379143714905
[8600] - Eval Loss -> 0.2151886373758316
[8650] - Loss -> 0.22001217305660248
[8700] - Loss -> 0.261572927236557
[8700] - Eval Loss -> 0.19051678478717804
[8750] - Loss -> 0.2679530382156372
[8800] - Loss -> 0.10333076864480972
[8800] - Eval Loss -> 0.3242172598838806
[8850] - Loss -> 0.174746572971344
[8900] - Loss -> 0.23519644141197205
[8900] - Eval Loss -> 0.44062504172325134
[8950] - Loss -> 0.15766175091266632
[9000] - Loss -> 0.20071816444396973
[9000] - Eval Loss -> 0.28983303904533386
[9050] - Loss -> 0.19144663214683533
[9100] - Loss -> 0.22883915901184082
[9100] - Eval Loss -> 0.29681888222694397
[9150] - Loss -> 0.2150583118200302
[9200] - Loss -> 0.21388939023017883
[9200] - Eval Loss -> 0.22537973523139954
[9250] - Loss -> 0.15630744397640228
```

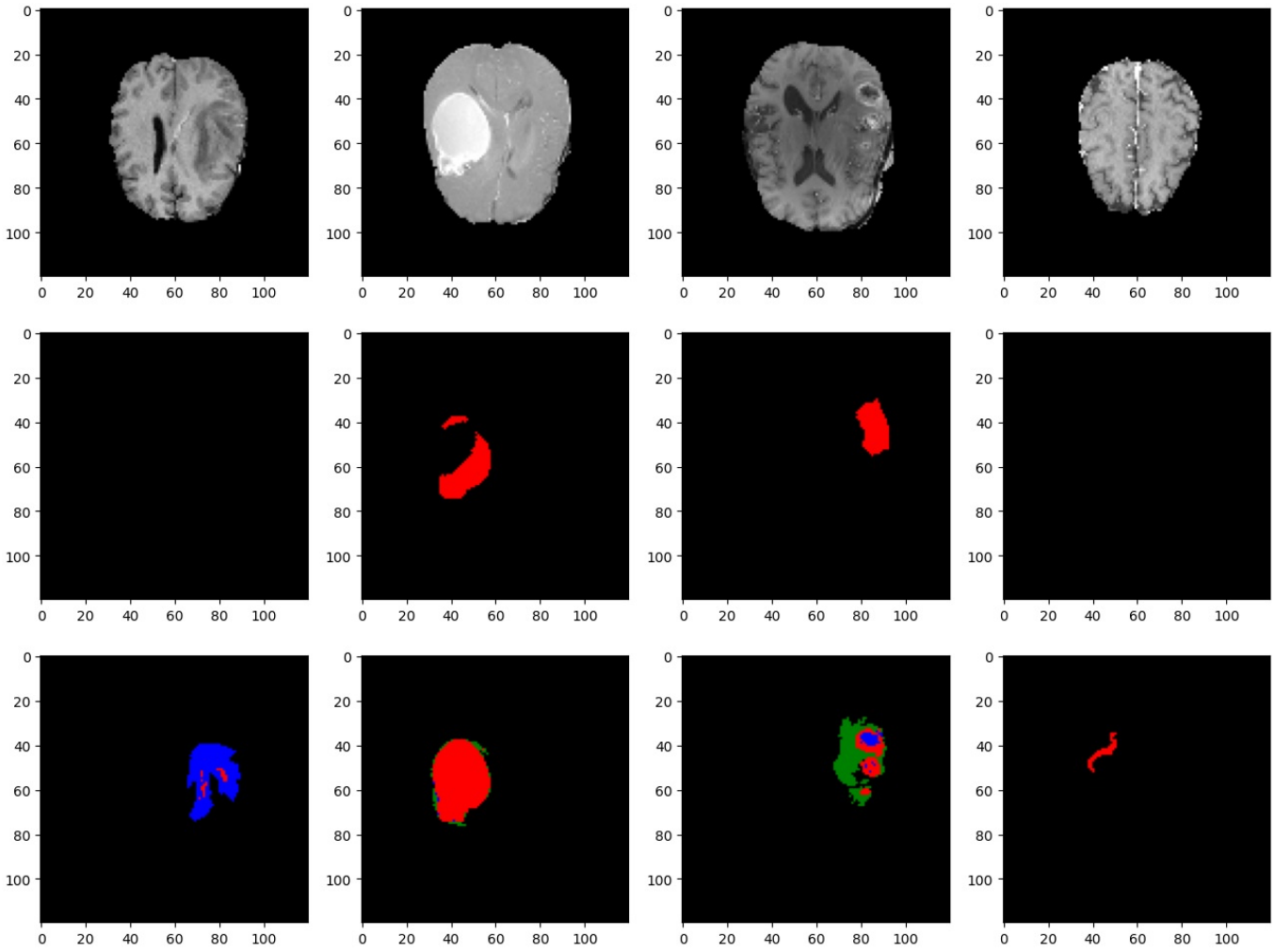
```
[9300] - Loss -> 0.22089211642742157
[9300] - Eval Loss -> 0.18903927505016327
[9350] - Loss -> 0.2000611424446106
[9400] - Loss -> 0.2476840317249298
[9400] - Eval Loss -> 0.4048024117946625
[9450] - Loss -> 0.2531484365463257
[9500] - Loss -> 0.19486716389656067
[9500] - Eval Loss -> 0.23484373092651367
[9550] - Loss -> 0.18327750265598297
[9600] - Loss -> 0.17621278762817383
[9600] - Eval Loss -> 0.2899583578109741
[9650] - Loss -> 0.25468504428863525
[9700] - Loss -> 0.2046280801296234
[9700] - Eval Loss -> 0.33225077390670776
[9750] - Loss -> 0.28684744238853455
[9800] - Loss -> 0.1878071427345276
[9800] - Eval Loss -> 0.23251140117645264
[9850] - Loss -> 0.2533099055290222
[9900] - Loss -> 0.18275462090969086
[9900] - Eval Loss -> 0.308765172958374
[9950] - Loss -> 0.21091464161872864
[10000] - Loss -> 0.2078237235546112
[10000] - Eval Loss -> 0.22486017644405365
Training took 493.948s 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 [185…

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

def categorise_segments(logit : torch.Tensor):
    return np.array(torch.max(logit, dim=0)[1].tolist())



num_examples = 4
model.eval()

images, labels = test_set.get_random_batch(num_examples)
tensor_images, tensor_labels = torch.from_numpy(images), torch.from_numpy(labels)
tensor_images, tensor_labels = tensor_images.to(device, dtype=torch.float32), tensor_labels.to(device, dtype=to
logits = model(tensor_images)

for i in range(num_examples):
    plt.subplot(3,num_examples,i+1)
    plt.imshow(images[i,0], cmap='gray')

    plt.subplot(3,num_examples,i+5)
    plt.imshow(categorise_segments(logits[i]), cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))

    plt.subplot(3,num_examples,i+9)
    plt.imshow(labels[i], cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
plt.gcf().set_size_inches(num_examples * 4, num_examples * 3)
### 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 partially works. It can identify with some success the enhancing tumours, but it doesn't seem to be able to identify the edema or non-enhancing tumours. It also doesn't seem to be able to detect tumours that are too small or thin. One of the things that could be done to improve this model, that would probably significantly help would be improving the model's ability to focus on specific features, such as edges within the images. Another thing that could help would be to augment the training set, by adding noise to the images, as well as different transformations, although the transformations may be less important with this dataset due to the fact that the brains are relatively consistently positioned within the images. Another thing that maybe could help would be if the model took into account knowledge about the fact that these scans are not all independent from each other, and in fact contains collections that are scans at different (known) vertical positions through the brain. I feel like structures like tumours may be easier to identify in 3d rather than 2d.