Coursework 2: Image segmentation

In this coursework you will develop and train a convolutional neural network for brain tumour image segmentation. Please read both the text and the code in this notebook to get an idea what you are expected to implement. Pay attention to the missing code blocks that look like this:

```
### Insert your code ###
...
### End of your code ###
```

What to do?

- Complete and run the code using `jupyter-lab` or `jupyter-notebook` to get the results.
- Export (File | Save and Export Notebook As...) the notebook as a PDF file, which contains your code, results and answers, and upload the PDF file onto Scientia.
- Instead of clicking the Export button, you can also run the following command instead: `jupyter nbconvert coursework.ipynb --to pdf`
- If Jupyter complains about some problems in exporting, it is likely that pandoc (https://pandoc.org/installing.html) or latex is not installed, or their paths have not been included. You can install the relevant libraries and retry.
- If Jupyter-lab does not work for you at the end, you can use Google Colab to write the code and export the PDF file.

Dependencies

You need to install Jupyter-Lab (https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html) and other libraries used in this coursework, such as by running the command: `pip3 install [package_name]`

GPU resource

The coursework is developed to be able to run on CPU, as all images have been pre-processed to be 2D and of a smaller size, compared to original 3D volumes.

However, to save training time, you may want to use GPU. In that case, you can run this notebook on Google Colab. On Google Colab, go to the menu, Runtime - Change runtime type, and select **GPU** as the hardware acceleartor. At the end, please still export everything and submit as a PDF file on Scientia.

In [ ]:
```
# Import libraries
# These libraries should be sufficient for this tutorial.
# However, if any other library is needed, please install by yourself.
import tarfile
import imageio
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
import numpy as np
import time
import os
import random
import matplotlib.pyplot as plt
from matplotlib import colors
```

1. Download and visualise the imaging dataset.

The dataset is curated from the brain imaging dataset in Medical Decathlon Challenge. To save the storage and reduce the computational cost for this tutorial, we extract 2D image slices from T1-Gd contrast enhanced 3D brain volumes and downsample the images.

The dataset consists of a training set and a test set. Each image is of dimension 120 x 120, with a corresponding label map of the same dimension. There are four number of classes in the label map:

- 0: background
- 1: edema
- 2: non-enhancing tumour
- 3: enhancing tumour

In [ ]:
```
# Download the dataset
!wget https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz

# Unzip the '.tar.gz' file to the current directory
datafile = tarfile.open('Task01_BrainTumour_2D.tar.gz')
datafile.extractall()
datafile.close()
```

```
--2023-03-03 17:24:58--  https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTum
our_2D.tar.gz
Resolving www.dropbox.com (www.dropbox.com)... 162.125.1.18, 2620:100:6016:18::a27
d:112
Connecting to www.dropbox.com (www.dropbox.com)|162.125.1.18|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz [following]
--2023-03-03 17:24:58--  https://www.dropbox.com/s/raw/zmytk2yu284af6t/Task01_Brai
nTumour_2D.tar.gz
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://uc4559ca937d6de3c9a226ca6047.dl.dropboxusercontent.com/cd/0/inli
ne/B3jzf-t_KtzSZ6i1JUSfDJqKPEjqkJWSBZjqEGRKw9OqAAjEwo-Oo_RkqbTvU8se-VINx0n5x62YsZc
Ak3PuPybM7veM9zJOzaQRCTwHCFf0B6oEKFkC8fXyO1YbjuaUULyRpfbsm8LqR_eLk7WSo6vIdI_3Z_LAX
8UV5rWYF-TVGQ/file# [following]
--2023-03-03 17:24:59--  https://uc4559ca937d6de3c9a226ca6047.dl.dropboxuserconten
t.com/cd/0/inline/B3jzf-t_KtzSZ6i1JUSfDJqKPEjqkJWSBZjqEGRKw9OqAAjEwo-Oo_RkqbTvU8se
-VINx0n5x62YsZcAk3PuPybM7veM9zJOzaQRCTwHCFf0B6oEKFkC8fXyO1YbjuaUULyRpfbsm8LqR_eLk7
WSo6vIdI_3Z_LAX8UV5rWYF-TVGQ/file
Resolving uc4559ca937d6de3c9a226ca6047.dl.dropboxusercontent.com (uc4559ca937d6de3
c9a226ca6047.dl.dropboxusercontent.com)... 162.125.1.15, 2620:100:6016:15::a27d:10
f
Connecting to uc4559ca937d6de3c9a226ca6047.dl.dropboxusercontent.com (uc4559ca937d
6de3c9a226ca6047.dl.dropboxusercontent.com)|162.125.1.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: /cd/0/inline2/B3gsm1Fbh4ZeQgqRrAErFWzrn7Z4KX9WqEpX1UOsK7DntnYjeoA3G9LCid
MtHd07YnGkhbzPk-cXkOhzwVAGIuQlmGWl9tehq5JJmSC8W-mqCp6aEgeQdr1Fh9fg963vhIPCloon4XNZ
8VelMQfrr5qVef7cbKjaimOuvv2KqmfL39Po_-zOOvXF5_r_A-vmLrC9L9vE9c4kxitsiXjIJVXmVg3rYQ
7Bewav-xbCs6l13uBJvQU6k91cUjHMyYkw8k0iK67F7i_A2-YhQvdjgU6UgxdfpsRXcwL96HWNRPWpXlfZ
5BC2LPLzCCR7ydExfZmaPrwozstx-qCG3mv88A1rVDOPhOBls6JsJlMPQSHJ6aRyOKvBuGDWbNbkDiQ4u8
02p61QSTnGm_0ZdK7ZJQxhjSyZ9wVcJnvJb3UWIwB2cQ/file [following]
--2023-03-03 17:24:59--  https://uc4559ca937d6de3c9a226ca6047.dl.dropboxuserconten
t.com/cd/0/inline2/B3gsm1Fbh4ZeQgqRrAErFWzrn7Z4KX9WqEpX1UOsK7DntnYjeoA3G9LCidMtHd0
7YnGkhbzPk-cXkOhzwVAGIuQlmGWl9tehq5JJmSC8W-mqCp6aEgeQdr1Fh9fg963vhIPCloon4XNZ8VelM
Qfrr5qVef7cbKjaimOuvv2KqmfL39Po_-zOOvXF5_r_A-vmLrC9L9vE9c4kxitsiXjIJVXmVg3rYQ7Bewa
v-xbCs6l13uBJvQU6k91cUjHMyYkw8k0iK67F7i_A2-YhQvdjgU6UgxdfpsRXcwL96HWNRPWpXlfZ5BC2L
PLzCCR7ydExfZmaPrwozstx-qCG3mv88A1rVDOPhOBls6JsJlMPQSHJ6aRyOKvBuGDWbNbkDiQ4u802p61
QSTnGm_0ZdK7ZJQxhjSyZ9wVcJnvJb3UWIwB2cQ/file
Reusing existing connection to uc4559ca937d6de3c9a226ca6047.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  40.7MB/s    in 0.2s

2023-03-03 17:25:00 (40.7 MB/s) - ‘Task01_BrainTumour_2D.tar.gz.1’ saved [9251149/
9251149]
```

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

Suggested colour map for brain MR image:

```
cmap = 'gray'
```

Suggested colour map for segmentation map:

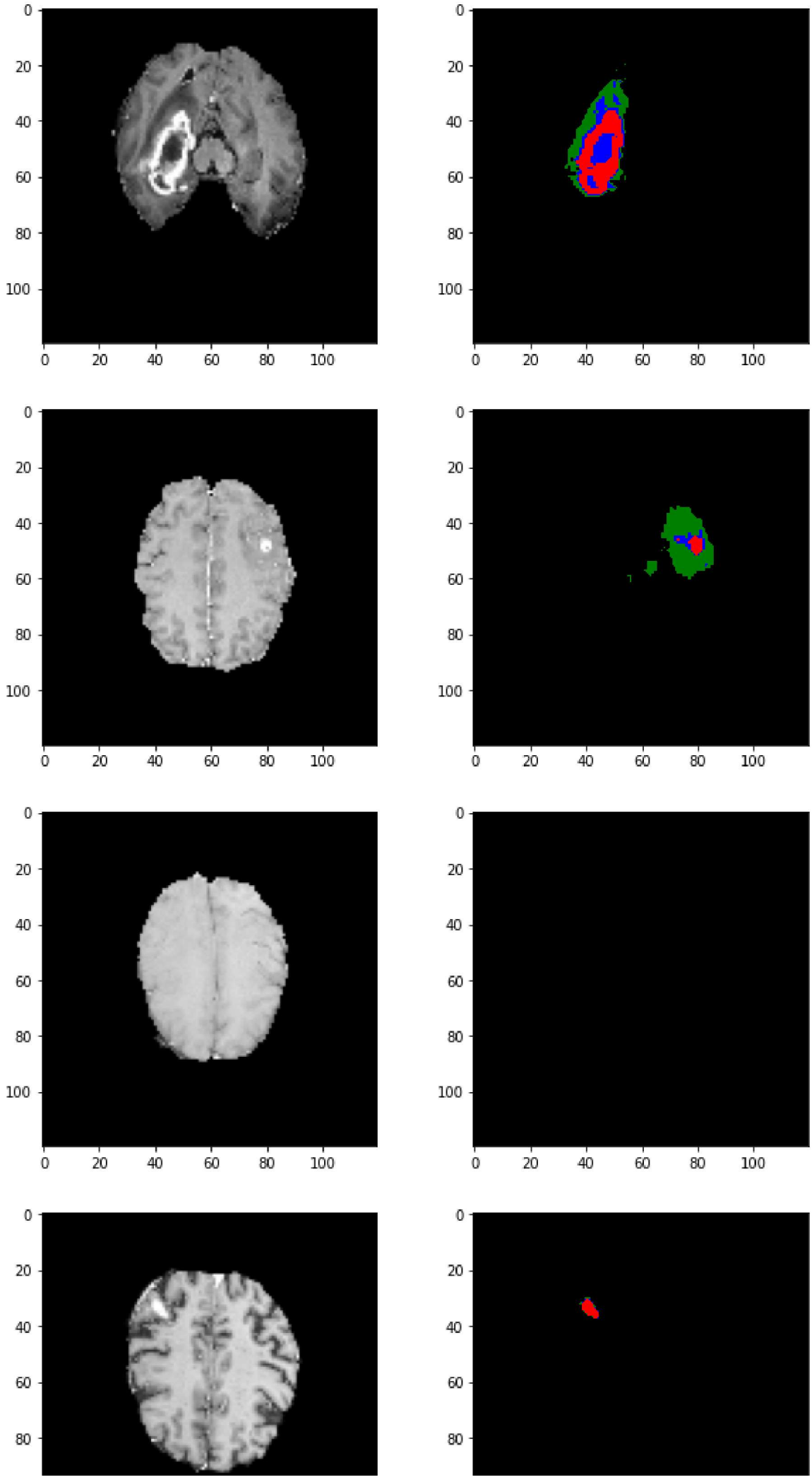
```
cmap = colors.ListedColormap(['black', 'green', 'blue', 'red'])
```

In [ ]:
```
### Insert your code ###
training_path = "./Task01_BrainTumour_2D/training_images/"
label_path = "./Task01_BrainTumour_2D/training_labels/"
imgs = os.listdir(training_path)
random_imgs = random.sample(imgs,4)

fig, axes = plt.subplots(4,2,figsize = (10,20))

# Loop through each image and display it in grayscale format
for i in range(4):
    image = imageio.imread(training_path + random_imgs[i], as_gray = True)
    label_image = imageio.imread(label_path + random_imgs[i], as_gray = True )
    axes[i,0].imshow(image, cmap='gray')
    axes[i,1].imshow(label_image, cmap=colors.ListedColormap(["black","green","blu

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



2. Implement a dataset class.

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

In [ ]:
```
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)

        images = np.array([[self.images[i]] for i in indices])
        labels = np.array([self.labels[i] for i in indices])
        return images, labels
```

3. Build a U-net architecture.

You will implement a U-net architecture. If you are not familiar with U-net, please read this paper:

[1] Olaf Ronneberger et al. U-Net: Convolutional networks for biomedical image segmentation. MICCAI, 2015.

For the first convolutional layer, you can start with 16 filters. We have implemented the encoder path. Please complete the decoder path.

In [ ]:
```
""" U-net """
class UNet(nn.Module):
    def __init__(self, input_channel=1, output_channel=1, num_filter=16):
        super(UNet, self).__init__()

        # BatchNorm: by default during training this layer keeps running estimates
        # of its computed mean and variance, which are then used for normalization
        # during evaluation.

        # Encoder path
        n = num_filter  # 16
        self.conv1 = nn.Sequential(
            nn.Conv2d(input_channel, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n *= 2  # 32
        self.conv2 = nn.Sequential(
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n *= 2  # 64
        self.conv3 = nn.Sequential(
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        n *= 2  # 128
        self.conv4 = nn.Sequential(
```

```
            nn.Conv2d(int(n / 2), n, kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU(),
            nn.Conv2d(n, n, kernel_size=3, padding=1),
            nn.BatchNorm2d(n),
            nn.ReLU()
        )

        # Decoder path
        ### Insert your code ###

        n//=2 # 64
        self.upconv4 = nn.ConvTranspose2d(n*2, n, kernel_size=2, stride=2)
        self.decode4 = 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//=2 #32
        self.upconv3 = nn.ConvTranspose2d(n*2, n, kernel_size=2, stride=2)
        self.decode3 = 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//=2 #16
        self.upconv2 = nn.ConvTranspose2d(n*2, n, kernel_size=2, stride=2)
        self.decode2 = 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.final = nn.ConvTranspose2d(num_filter, output_channel, kernel_size=1)
        ### End of your code ###

    def forward(self, x):
        # Use the convolutional operators defined above to build the U-net
        # The encoder part is already done for you.
        # You need to complete the decoder part.
        # Encoder
        x = self.conv1(x)
        conv1_skip = x

        x = self.conv2(x)
        conv2_skip = x

        x = self.conv3(x)
        conv3_skip = x

        x = self.conv4(x)

        # Decoder
```

```
        ### Insert your code ###
        x = self.upconv4(x)
        x = torch.cat((x, conv3_skip), dim=1)
        x = self.decode4(x)

        x = self.upconv3(x)
        x = torch.cat((x, conv2_skip), dim=1)
        x = self.decode3(x)

        x = self.upconv2(x)
        x = torch.cat((x, conv1_skip), dim=1)
        x = self.decode2(x)

        x = self.final(x)
        ### End of your code ###
        return x
```

4. Train the segmentation model.

In [ ]:

```
# CUDA device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Device: {0}'.format(device))

# Build the model
num_class = 4
model = UNet(input_channel=1, output_channel=num_class, num_filter=16)
model = model.to(device)
params = list(model.parameters())

model_dir = 'saved_models'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

# Optimizer
optimizer = optim.Adam(params, lr=1e-3)

# Segmentation loss
criterion = nn.CrossEntropyLoss()

# Datasets
train_set = BrainImageSet('Task01_BrainTumour_2D/training_images', 'Task01_BrainTu
test_set = BrainImageSet('Task01_BrainTumour_2D/test_images', 'Task01_BrainTumour_

# Train the model
# Note: when you debug the model, you may reduce the number of iterations or batch
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,
    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, dty
    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()
    ### End of your code ###
    # Evaluate
    if it % 100 == 0:
        model.eval()
        # Disabling gradient calculation during reference to reduce memory consumpt
        with torch.no_grad():
            images, labels = test_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(dev
            logits = model(images)
            print("Loss: ",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'.forma
print('Training took {:.3f}s in total.'.format(time.time() - start))
```

```
Device: cuda
Loss:  0.2640340030193329
Loss:  0.14485397934913635
Loss:  0.12993885576725006
Loss:  0.09503228217363358
Loss:  0.11077208071947098
Loss:  0.05931940674781799
Loss:  0.06851973384618759
Loss:  0.08747299015522003
Loss:  0.04566098377108574
Loss:  0.05385855957865715
Loss:  0.024874825030565262
Loss:  0.04340526834130287
Loss:  0.04510243237018585
Loss:  0.04945544898509979
Loss:  0.05136760696768761
Loss:  0.04346807673573494
Loss:  0.031157052144408226
Loss:  0.039043281227350235
Loss:  0.04748892784118652
Loss:  0.027678698301315308
Loss:  0.0330725722014904
Loss:  0.039579667150974274
Loss:  0.021864548325538635
Loss:  0.02622385323047638
Loss:  0.032913461327552795
Loss:  0.024325324222445488
Loss:  0.02040429227054119
Loss:  0.02818884886801243
Loss:  0.019587071612477303
Loss:  0.013838930055499077
Loss:  0.01549094170331955
Loss:  0.024966640397906303
Loss:  0.02158743515610695
Loss:  0.022549239918589592
Loss:  0.01660737581551075
Loss:  0.020089905709028244
Loss:  0.015912514179944992
Loss:  0.020818401128053665
Loss:  0.016468193382024765
Loss:  0.014908282086253166
Loss:  0.013930842280387878
Loss:  0.015843462198972702
Loss:  0.027116697281599045
Loss:  0.009761642664670944
Loss:  0.01513082068413496
Loss:  0.011635737493634224
Loss:  0.015239743515849113
Loss:  0.013164238072931767
Loss:  0.014434709213674068
Loss:  0.009277192875742912
Loss:  0.016765372827649117
Loss:  0.008867695927619934
Loss:  0.013004598207771778
Loss:  0.017122842371463776
Loss:  0.008085005916655064
Loss:  0.0176344346255064
Loss:  0.010713525116443634
Loss:  0.008226323872804642
Loss:  0.011580702848732471
Loss:  0.008075675927102566
Loss:  0.009388417936861515
Loss:  0.010460474528372288
Loss:  0.009447631426155567
```

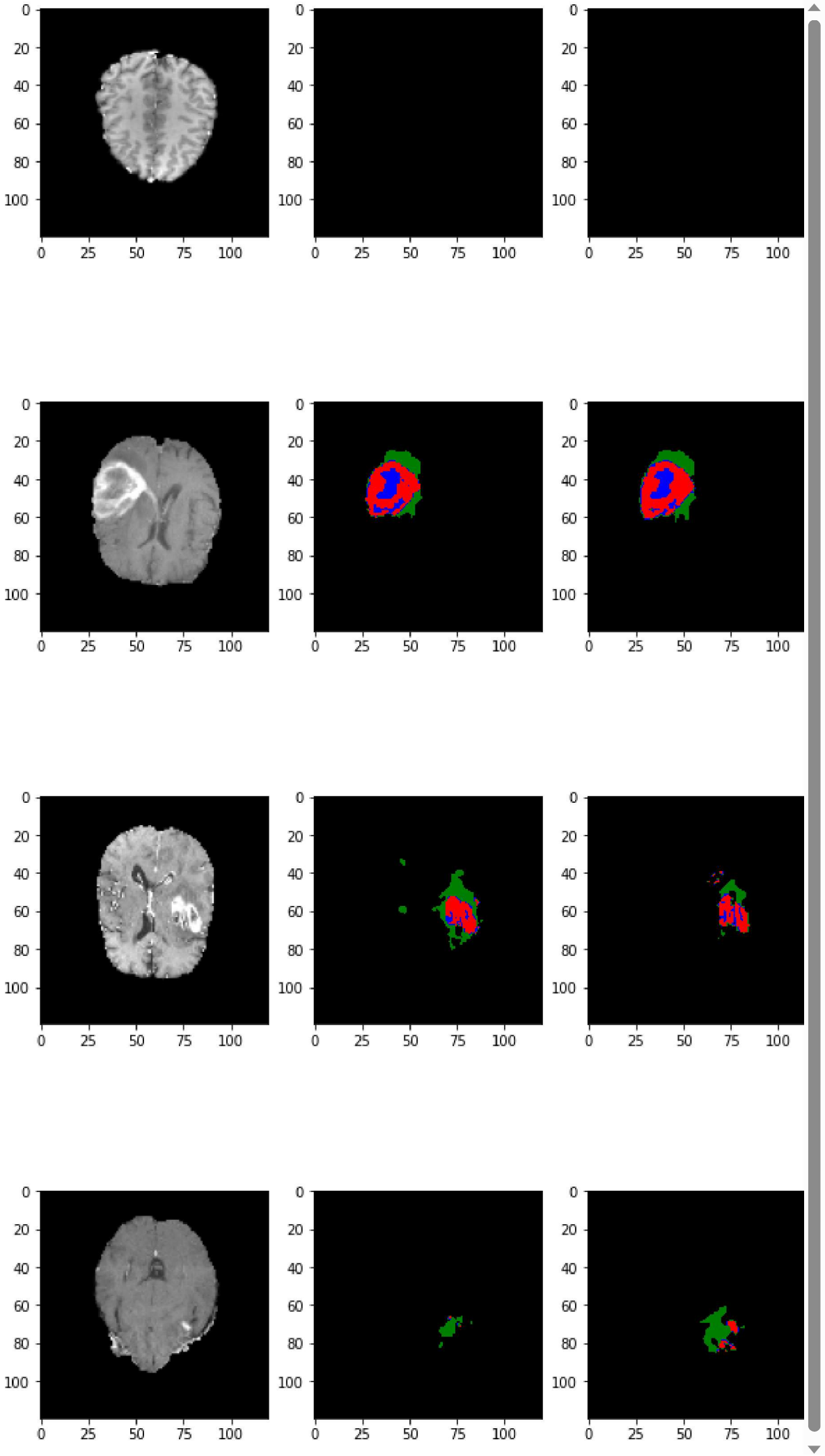
```
Loss:  0.011255228891968727
Loss:  0.013010196387767792
Loss:  0.013595865108072758
Loss:  0.01210114173591137
Loss:  0.014378946274518967
Loss:  0.009476935490965843
Loss:  0.012689425610005856
Loss:  0.009150156751275063
Loss:  0.011801484040915966
Loss:  0.013229694217443466
Loss:  0.011221745982766151
Loss:  0.011785130947828293
Loss:  0.009404107928276062
Loss:  0.01379519421607256
Loss:  0.011934112757444382
Loss:  0.00612838426604867
Loss:  0.004508203826844692
Loss:  0.011427056975662708
Loss:  0.010907597839832306
Loss:  0.013295433484017849
Loss:  0.012776512652635574
Loss:  0.010439801029860973
Loss:  0.010534144937992096
Loss:  0.008729771710932255
Loss:  0.009190146811306477
Loss:  0.01148657500743866
Loss:  0.006771838292479515
Loss:  0.006817578803747892
Loss:  0.010925288312137127
Loss:  0.008128994144499302
Loss:  0.007305378559976816
Loss:  0.009492088109254837
Loss:  0.006544598378241062
Loss:  0.008838759735226631
Loss:  0.011825613677501678
Loss:  0.009757347404956818
Loss:  0.00531874131411314
Training took 356.666s in total.
```

5. Deploy the trained model to a random set of 4 test images and visualise the automated segmentation.

You can show the images as a 4 x 3 panel. Each row shows one example, with the 3 columns being the test image, automated segmentation and ground truth segmentation.

In [ ]:
```
### Insert your code ###
model_path = "./saved_models/model_10000.pt"
model.load_state_dict(torch.load(model_path))
fig, axes = plt.subplots(4,3,figsize = (10,20))
test_image_path = "./Task01_BrainTumour_2D/test_images/"
test_label_path = "./Task01_BrainTumour_2D/test_labels/"
imgs = os.listdir(test_image_path)
random_imgs = random.sample(imgs,4)
for i in range(4):
    original_image = imageio.imread(test_image_path + random_imgs[i], as_gray = Tr
    image = np.expand_dims(original_image, axis=0)
    image = np.expand_dims(image, axis=1)
    image = torch.from_numpy(image).to(device, dtype=torch.float32)
    model.eval()
    with torch.no_grad():
```

```
        logits = model(image)
        pred = logits.argmax(dim=1).squeeze().cpu().numpy()
    label_image = imageio.imread(test_label_path + random_imgs[i], as_gray = True)
    axes[i,0].imshow(original_image, cmap= 'gray')
    axes[i,1].imshow(pred, cmap= colors.ListedColormap(["black","green","blue","re
    axes[i,2].imshow(label_image, cmap= colors.ListedColormap(["black","green","bl
### End of your code ###
```


0 20 40 60 80 100
0 25 50 75 100

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

As seen by the loss values (final loss of 0.0053) and the visual representation above, the results of the model compared to the ground truth are in most cases accurate enough to identify the general region of the tumour, but the segmentation is not accurate enough to outline it completely. This is especially apparent in the fourth example above, where the ground truth segment is larger than the predicted region. This loss could cause false positives and negatives so it will need to be improved before being deployed to a real clinic. This can be done in several ways:

UNet++ is an improved model of the current U-Net architecture. It uses a nested and dense skip connection network which allows it to capture features at different scales and combine them. This is done through an increase in convolutional layers that concatenate feature maps from the earlier layers.

UNETR is a model that uses a transformer architecture. This allows it to capture not only local features learnt by the convolutions, but also global multi-scale features learnt by the transformer, which results in enhanced feature processing and more information captured from input images.

UHR-Net is another improvement on the U-Net model that allows better handling of high-resolution images. It uses pyramid pooling to capture multi-scale contextual information from input images. In addition to this, it contains more skip connections and convolution layers to prevent information loss during encoding and decoding. This is very important as our current model lacks accuracy when tracing tumours which can be due to a higher information loss with high resolution images.

Finally, dilated convolution can be used to increase context information (from the input images) while maintaining a high resolution of feature maps. This can also reduce information loss during pooling. This is done by inserting spaces between the kernel elements by using a dilation rate parameter. This increases the receptive field of the kernel without increasing its size or number of parameters and increases stride in the convolution operation.

Features from these methods and architectures can be combined to increase the accuracy of this U-Net segmentation model before it can be used in a real clinic.