coursework-02-solutions

February 21, 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.

```
[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
import imageio.v2 as imageio
```

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]: # 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()
```

```
--2024-02-20 22:47:36--
https://www.dropbox.com/s/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.g
z
Resolving www.dropbox.com (www.dropbox.com)… 162.125.72.18
Connecting to www.dropbox.com (www.dropbox.com)|162.125.72.18|:443…
connected. HTTP request sent, awaiting response… 302 Found
Location: /s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.tar.gz
[following]
--2024-02-20 22:47:37--
```

```
https://www.dropbox.com/s/raw/zmytk2yu284af6t/Task01_BrainTumour_2D.t
ar.gz
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response… 302 Found
Location:
https://uc753aac83ec967af22fd76014a8.dl.dropboxusercontent.com/cd/0/i
n
line/CNqtXp5ZFdkHhSYSmi6jrRmCveTQELYQMdTA1ha0RggFxhSH_Skt_D79nKQYPM0d
DKZ5q5YPd0n
Ymxd9iKnAP2kK_nmWdM_gpBttaoZ3dph2bJ6gmwKtd8d7BdcVD6yQdF0/file#
[following]
--2024-02-20                                                 22:47:38--
https://uc753aac83ec967af22fd76014a8.dl.dropboxusercont
ent.com/cd/0/inline/CNqtXp5ZFdkHhSYSmi6jrRmCveTQELYQMdTA1ha0RggFxhSH_
Skt_D79nKQY
PM0dDKZ5q5YPd0nYmxd9iKnAP2kK_nmWdM_gpBttaoZ3dph2bJ6gmwKtd8d7BdcVD6yQd
F0/file
Resolving uc753aac83ec967af22fd76014a8.dl.dropboxusercontent.com
(uc753aac83ec967af22fd76014a8.dl.dropboxusercontent.com)…
162.125.72.15
Connecting to uc753aac83ec967af22fd76014a8.dl.dropboxusercontent.com
(uc753aac83ec967af22fd76014a8.dl.dropboxusercontent.com)|162.125.72.15|:44
3… connected.
HTTP request sent, awaiting response… 302 Found
Location:     /cd/0/inline2/CNoU7AKqhfR9dfVfsoIhKttmMasxiVGYxmvTh-pq-
9WIZ95dBQhrDwZb
H5cn3yPdm93b7ld86fNlJWBID-
y2qOR4Fa9xV14liuAHyAq_HvsF8hKwWzcK1UDtUDfJcCSqdqpVCdK3
CX_7zZRn4MK8et3BQJDqeJ1ErOBhZrMC_ltS8MT3GxLh9MtASIrttx8Oi3POPA_Z9M9fTqc84v
JsS1Pf tIdGzxK42PROrMrYiXb8zudSpHqCN3Yq6E218rlGPFI5YHaiYRIp-
nxroRSDTYu1N-
Y76fq5MQmoZB5XZjqpT8wxfW0BcCcv9RTOMXg2KZnVOezPBjyhbZLwtCDoInjK-
YzvHnB92eUPOvqnNqQKsQ/file [following]
--2024-02-20                                                 22:47:39--
https://uc753aac83ec967af22fd76014a8.dl.dropboxusercont
ent.com/cd/0/inline2/CNoU7AKqhfR9dfVfsoIhKttmMasxiVGYxmvTh-pq-
9WIZ95dBQhrDwZbH5c                            n3yPdm93b7ld86fNlJWBID-
y2qOR4Fa9xV14liuAHyAq_HvsF8hKwWzcK1UDtUDfJcCSqdqpVCdK3CX_
7zZRn4MK8et3BQJDqeJ1ErOBhZrMC_ltS8MT3GxLh9MtASIrttx8Oi3POPA_Z9M9fTqc8
4vJsS1PftId
GzxK42PROrMrYiXb8zudSpHqCN3Yq6E218rlGPFI5YHaiYRIp-
nxroRSDTYu1N-
Y76fq5MQmoZB5XZjqpT8wxfW0BcCcv9RTOMXg2KZnVOezPBjyhbZLwtCDoInjK-
YzvHnB92eUPOvqnNqQKsQ/file Reusing existing
connection to
uc753aac83ec967af22fd76014a8.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.2'

Task01_BrainTumour_ 100%[===================>]8.82M 2.68MB/s  in 3.3s

2024-02-20 22:47:42 (2.68 MB/s) - 'Task01_BrainTumour_2D.tar.gz.2'
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'])`

[4]:
```
data_dir = './Task01_BrainTumour_2D/'

images_dir = os.path.join(data_dir, 'training_images')
labels_dir = os.path.join(data_dir, 'training_labels')
images = sorted(os.listdir(images_dir))
labels = sorted(os.listdir(labels_dir))

random_indices = random.sample(range(len(images)), 4)

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

cmap_img = 'gray'
cmap_label = colors.ListedColormap(['black', 'green', 'blue', 'red'])

for i, idx in enumerate(random_indices):
    img_path = os.path.join(images_dir, images[idx])
    label_path = os.path.join(labels_dir, labels[idx])

    img = imageio.imread(img_path)
    label = imageio.imread(label_path)

    axs[i, 0].imshow(img, cmap=cmap_img)
    axs[i, 0].set_title('Image')

    axs[i, 1].imshow(label, cmap=cmap_label, vmin=0, vmax=3)
    axs[i, 1].set_title('Label')

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

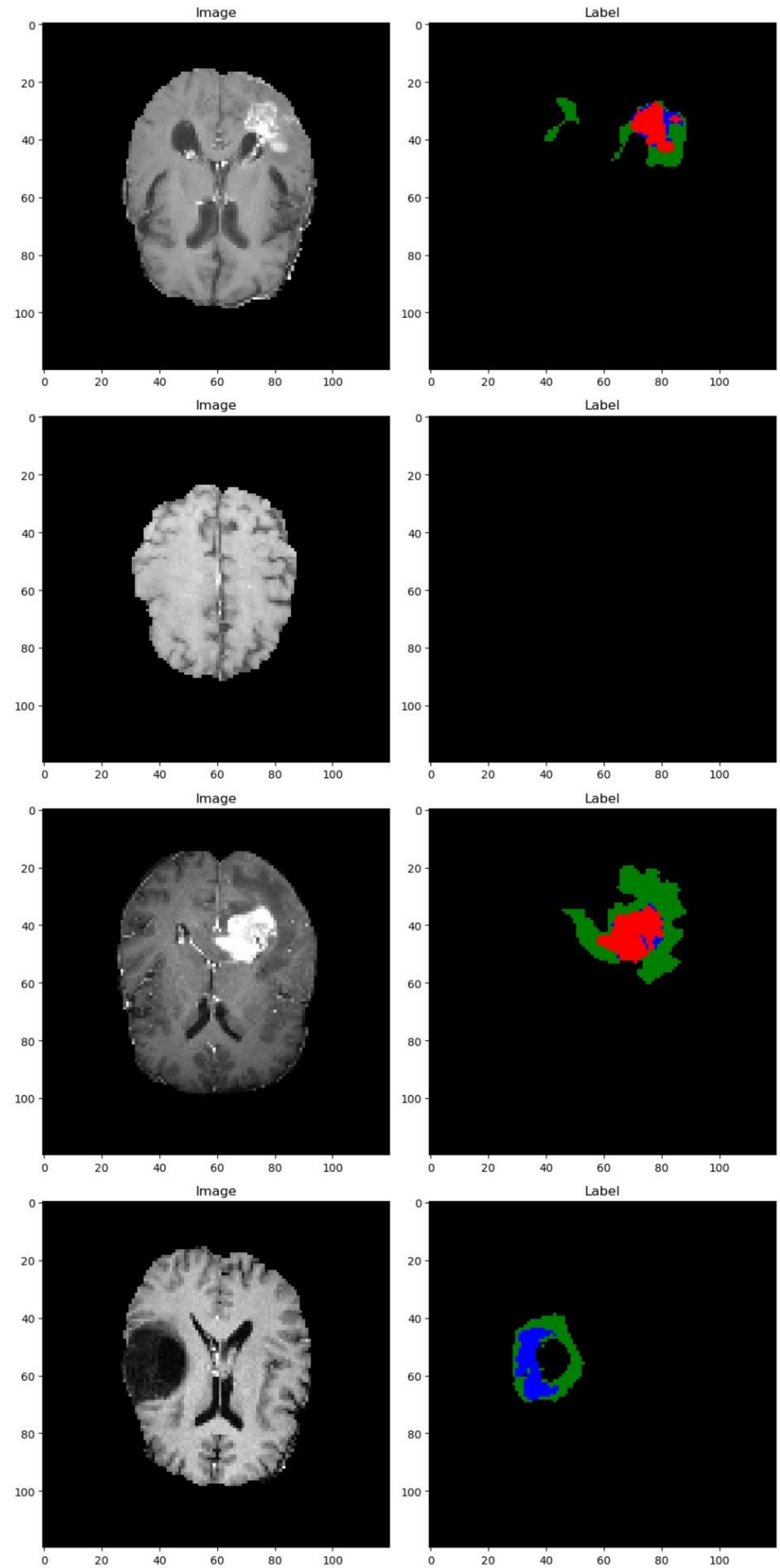


Image
Label
0
20
40
60
80
100
0
20
40
60
80
100
Image
Label
Image
Label
Image
Label

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.

[5]:
```
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
        #set empty array
        images, labels = [], []
        #take random number of indices
        indices = random.sample(range(self.__len__()), batch_size)
        #load random batch
        for idx in indices:
            image, label = self.__getitem__(idx)
            images.append(image)
            labels.append(label)
        images = np.array(images)
        labels = np.array(labels)
        #reshape image array to required shape
        new_images = np.reshape(images, (images.shape[0], 1, images.shape[1],␣
↪images.shape[2]))
        return new_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.

[6]:
```
""" 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()
        )

        #bottleneck
        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()
        )
        #upsample using transpose convolution
        self.upconv1 = nn.ConvTranspose2d(n, int(n/2), kernel_size=2, stride=2)
        # Decoder path
        n = int(n/2)
        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()
        )
        self.upconv2 = nn.ConvTranspose2d(n, int(n/2), kernel_size=2, stride=2)
        n = int(n/2)
        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()
        )
        self.upconv3 = nn.ConvTranspose2d(n, int(n/2), kernel_size=2, stride=2)
        n = int(n/2)
        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()
        )
        #final convolution of kernel size 1
        self.conv8 = nn.Conv2d(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)
        x = self.upconv1(x)

         #Decoder
        #concatonte the new x with the correspondihng encoder x (unet␣
↪architecture model)
        x = self.conv5(torch.cat([conv3_skip,x], dim = 1))
        x = self.upconv2(x)
```

```
        x = self.conv6(torch.cat([conv2_skip,x], dim = 1))
        x = self.upconv3(x)

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

        x = self.conv8(x)
        return x
```

1.8 4. Train the segmentation model.

[7]:
```
# mps device (fastest)
device = torch.device('mps')
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()

# Load 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 = 5000
# num_iter = 5000 because I found 10000 was overtraining

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)
    print(loss)
    loss.backward()
    optimizer.step()
    ### 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(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)
            loss = criterion(logits,labels)
            print(f'loss is {loss}')
            ### 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: mps
```

```
. . .

grad_fn=<NllLoss2DBackward0>)          tensor(0.0105,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0211,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0177,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0143,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0162,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0161,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0149,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0179,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0179,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0084,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0198,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0185,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0123,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0152,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0124,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0171,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0129,
device='mps:0',        grad_fn=<NllLoss2DBackward0>)
tensor(0.0151,                          device='mps:0',
grad_fn=<NllLoss2DBackward0>)          tensor(0.0138,
device='mps:0',  grad_fn=<NllLoss2DBackward0>)  loss
is  0.032718829810619354  Training  took  244.866s  in
total.
```

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

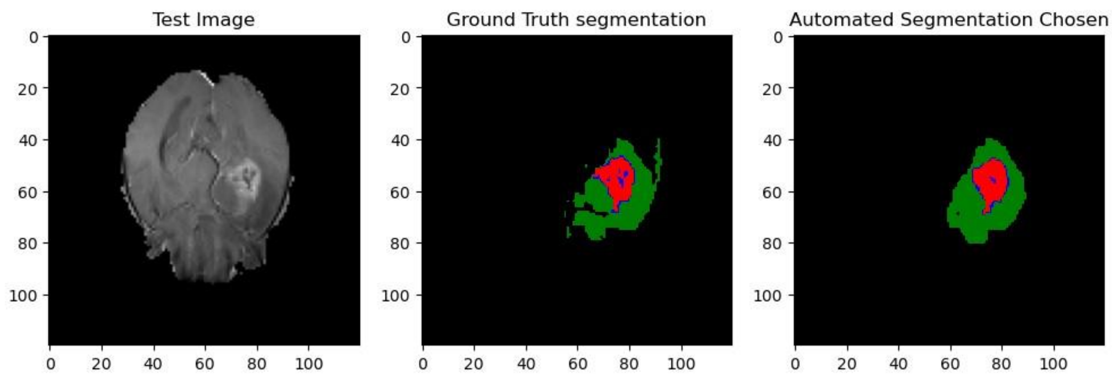
You can show the images as a 4 x 3 panel. Each row shows one example, with the 3 columns being the test image, automated segmentation and ground truth segmentation.

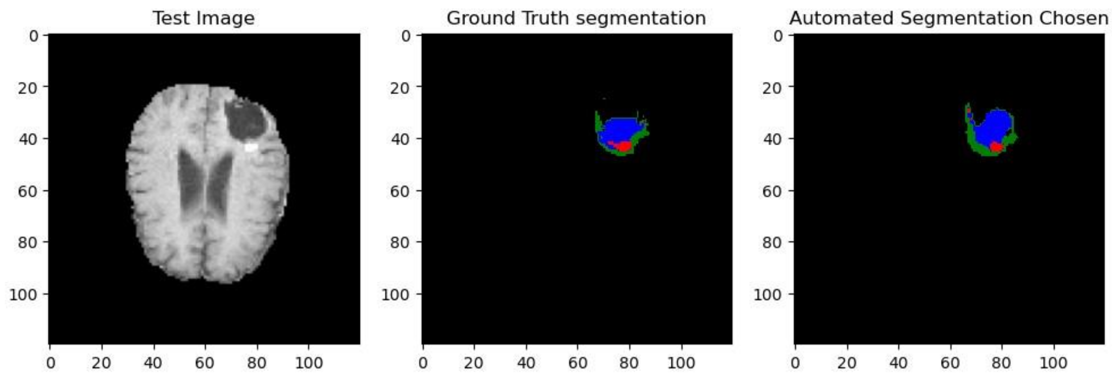
[13]:
```
### Insert your code ### #load
trained images model_path =
"saved_models/model_5000.pt"
# Instantiate an instance of the UNet model class with specific
parameters.
test_model = UNet(input_channel=1, output_channel=4,
num_filter=16) checkpoint = torch.load(model_path,
map_location=device)
```

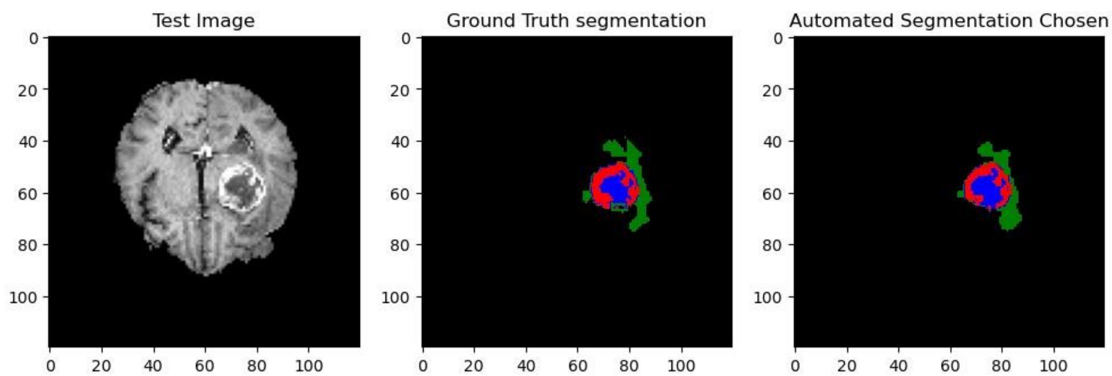
```
test_model.load_state_dict(checkpoint)
test_model.to(device)
#set the model to evaluation mode.This disables certain operations like
dropout.
test_model.eval()
test_model.eval()
#get random batch of 4 images from test set images, labels =
test_set.get_random_batch(4) images, labels =
torch.from_numpy(images).to(device, dtype=torch.float32),␣
↪torch.from_numpy(labels).to(device, dtype=torch.long)
#get raw output of images
logits = test_model(images)
loss = criterion(logits,
labels)
# print(f'type: {type(logits)}, {logits.shape}')

print(logits[0].shape
) print(labels.shape)
print(images.shape)
images = images.cpu()
# Reshape the images tensor to have the shape (batch_size, height,
 width)␣ ↪removing the channel dimension.
images=np.reshape(images, (images.shape[0], images.shape[2],
images.shape[3])) logits=logits.cpu() logits =
logits.detach().numpy() labels = labels.cpu()
```

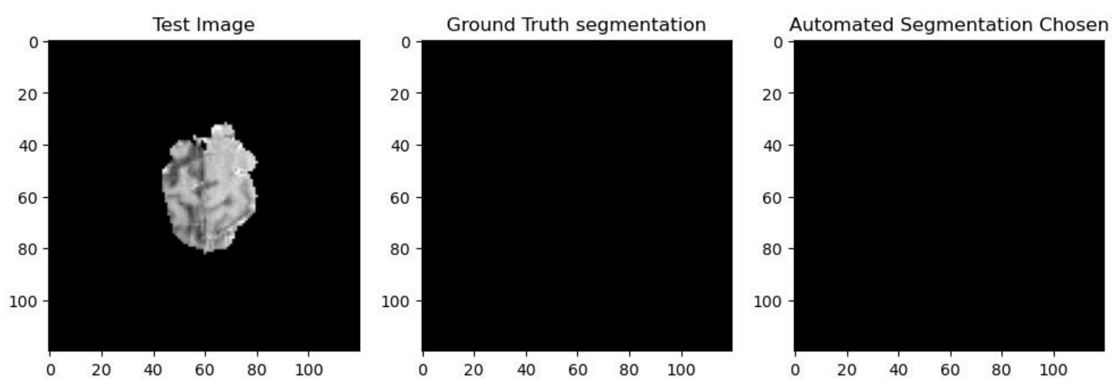
```python
# Output of CNN has shape NxCxHxW where N = batch size C = Number of classes and so on  we need to iterate though each image in the batch and sum all the classes to form 1 image

for i in range(4):
    plt.subplot(1, 3, 1)
    plt.title('Test Image')
    plt.imshow(images[i],cmap='gray')
    plt.subplot(1, 3, 2)
    plt.title('Ground Truth segmentation')
    plt.imshow(labels[i],cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
    plt.subplot(1, 3, 3)
    # Plot the automated segmentation chosen (argmax over the classes).
    plt.title('Automated Segmentation Chosen')
    plt.imshow(np.argmax(logits[i],axis=0),cmap = colors.ListedColormap(['black', 'green', 'blue', 'red']))
    # Set the size of the figure.
    plt.gcf().set_size_inches(12, 4)

    plt.show()
print(f'loss is {loss}')

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

```
torch.Size([4, 120, 120])
torch.Size([4,      120,      120])
torch.Size([4, 1, 120, 120])
```










```
loss is 0.02548111416399479
```

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?

-Model took roughly 4 mins to execute therefore was quite quick using mps. Model loss is 0.025, when I ran 10000 iterations it was around 0.3 therefore accuracy is a lost better. This was using cross entropy loss function, to further confirm that loss is low, and accuracy is high I can try evaluating with another loss criterion.

-The test images successfully identify different tumours of the brain using the same classes but different methods, eg. argmax for automated segmentation, therefore the training of the neural network seems successful.

-When I displayed the images at the start, in task 1, the colour maps are similar to after I trained the model but now on the test images, there is more accuracy and separating of the tumours on the new colour maps. You can also see that for healthy brains, no tumours are displayed in the colour map which indicates it is detecting tumours correctly.

-To improve I would consider perfecting the learning rate and weight of the images. I would also train my model on a different dataset before deployment to ensure my neural network has not overfitted to this specific dataset.